HOCHSCHULE
MITTWEIDA
UNIVERSITY OF
APPLIED SCIENCES

BACHELORARBEIT

Frau
Marika Obst

Realisierbarkeit
eines digitalen Films
basierend auf Games Engines

2012

Fakultät Medien

BACHELORARBEIT

Doppelarbeit

Realisierbarkeit eines digitalen Films basierend auf Games Engines

Autor:
Marika Obst und Eric Schubert

Studiengang:
Medientechnik

Seminargruppe:
MT08wF-B

Erstprüfer:
Prof. Dr.-Ing. Robert J. Wierzbicki

Zweitprüfer:
Dipl.-Ing. Sieglinde Klimant

Einreichung:
Mittweida, 16.01.2012

Faculty of Media

BACHELOR THESIS

Animated Movies Based on Games Engine Technology - A Feasibility Study

author:
Marika Obst and Eric Schubert

course of studies:
Media Technology

seminar group:
MT08wF-B

first examiner:
Prof. Dr.-Ing. Prof. Robert J. Wierzbicki

second examiner:
Dipl.-Ing. Sieglinde Klimant

submission:
Mittweida, 2012-01-16

Bibliografische Angaben:

Nachname, Vorname: Obst, Marika (Texte von Eric Schubert sind grau markiert)

Realisierbarkeit eines digitalen Films basierend auf Games Engines

Animated Movies Based on Games Engine Technology – A Feasibility Study

2012 - 107 Seiten

Mittweida, Hochschule Mittweida (FH), University of Applied Sciences,

Fakultät Medien, Bachelorarbeit, 2012

Abstract

Diese Bachelorarbeit thematisiert die Erstellung von 3D-animierten Filmen mit Games Engine Technologie. Diese Entwicklungsumgebung bietet potentiell gute, bisher nicht genutzte Möglichkeiten, filmische Werke zu kreieren.

Im Rahmen einer Machbarkeitsstudie behandelt diese Bachelorarbeit die Erstellung eines animierten Kurzfilms mit der *Unreal Engine 3*. Von der Content-Generierung bis zur finalen Ausgabe des Films wird jeder Arbeitsschritt betrachtet und auf seine Umsetzbarkeit in der Engine analysiert. Probleme werden explizit dargestellt und Lösungsansätze werden entwickelt. Auch inhaltliche Einschränkungen und Möglichkeiten werden im Bereich der Interaktivität und medienkonvergenten Unterhaltung aufgezeigt.

Am Schluss erfolgt die Auswertung des filmischen Ergebnisses, sowie ein Qualitätsvergleich zu bisherigen Arbeitweisen.

Inhaltsverzeichnis

Abkürzungsverzeichnis

ASE
… Ascii Scene Exporter

AVI
… Audio Video Interleave

BSP
… Binary Space Partition

CSG
… Constructive Solid Geometry

FOV
… Field of View

TGA
… Targa

UDK
… Unreal Development Kit

UU
… Unreal Unit

Formelverzeichnis

Kameraoptik

α	Field of View	[°]
d	Film- bzw. Sensorbreite	[m]
f	Brennweite	[m]

Abbildungsverzeichnis

Tabellenverzeichnis

1 Einleitung

3D-Visualisierungen werden heute zunehmend in allen Medienkanälen eingesetzt. Einerseits kann man Virtuelles verbildlichen und damit neue Welten kreieren, aber auch die bestehende Welt abbilden. Einsatzbereiche für 3D-Animationen und -Modelle sind vielfältig. So gibt es unzählige Filme, die komplett computergeneriert sind, sowie Produktvisualisierungen. Aber auch Architektur wird mit 3D-Modellen erlebbar, bevor auch nur ein Stein auf den anderen gesetzt wird. Wie überall ist der Zeitdruck auch bei Medienproduktionen hoch. Die Zeitspanne zwischen Auftrag und Deadline wird immer geringer. Deshalb ist es nötig, effiziente und unkonventionelle Wege für die Erstellung von digitalen Filmen zu gehen.

Eine Spiele-Engine (englisch: Games Engine (deutsch etwa: Spiel-Motor)) ist eine spezielle Laufzeitumgebung für Computerspiele, welche den Spielverlauf steuert und für die visuelle Darstellung des Spielablaufs verantwortlich ist.[1] In der Regel werden derartige Plattformen auch als Entwicklungsumgebung genutzt und bringen dafür auch die nötigen Werkzeuge mit. Eine Games Engine wird bisher meist nur für die Erstellung von Videospielen eingesetzt. Dabei sind die Möglichkeiten des Echtzeit-Renderings, welches Games Engines ermöglichen, wesentlich vielfältiger.

Der 3D-Animationsfilm löst den 2D-Zeichentrickfilm zusehends ab. So sind Pixars Animationsfilme, wie zum Beispiel „Oben“[2], im Wesentlichen erfolgreicher als die gezeichneten Filme der Disney Animationsstudios nach 2007. So hatte der 2009 erschienene Zeichentrickfilm „Küss den Frosch“ innerhalb von sechs Wochen ca. 880.000 Besucher.[3] Der 3D-Animationsfilm „Oben“ aus dem Hause Pixar hatte in der gleichen Zeit knapp 2 Millionen Besucher.[4] Doch der konventionelle 3D-Animationsfilm ist teuer. Als konventionell werden hier Filme verstanden, welche übliche Produktionswege gehen. So hat „Oben“ ein Budget von 105 Millionen US-Dollar verbraucht.[4] 3D-Animationsfilme sind teuer und aufwändig zu realisieren. Vieles muss von Hand erstellt werden, ein Stab von Animatoren und 3D-Künstler bewegt jedes Detail, das später im Bild zu sehen sein soll. Dies wird zwar zusehends durch 3D-Barbeitungsprogramme vereinfacht,

1 Vgl. Games Engine Architecture, Jason Gregory, Seite 3

2 http://www.imdb.de/title/tt1049413/ (Entnahmedatum: 13.12.2011)

3 http://www.zelluloid.de/filme/details.php3?id=13021 (Entnahmedatum: 11.12.2011)

4 http://www.zelluloid.de/filme/details.php3?id=19226 (Entnahmedatum: 11.12.2011)

verlangt jedoch immer noch viel Erfahrung und Zeitaufwand. Allerdings werden für Serienformate, die preisgünstig zu produzieren sind, Arbeitsabläufe benötigt, die ein schnelles und gleichzeitig optisch akzeptables Ergebnis garantieren. Um bei der Produktion von Filmen wirtschaftlich zu bleiben, müssen die Kosten für das Personal niedrig gehalten werden. Es ist kaum möglich, einen Stab von Animatoren, 3D-Künstlern und Technikern zu beschäftigen. Das Generieren von dreidimensionalen Bildern mit Hilfe von Computern (englisch: Rendering) in Echtzeit und vereinfachte Arbeitsabläufe würden hier Kosten und Zeit sparen. Der Prozess des Echtzeit-Renderings ist die Grundlage für Videospiele. Jedoch sind Animationssequenzen in Videospielen eher Mittel zum Zweck. Dem Spieler wird in kurzen Sequenzen eine Ausgangslage oder eine Wirkung seines Handels gezeigt.

Ein filmisches Produkt aus Spielen sind sogenannte Machinimas[5], also Filme, die in einer Games Engine erzeugt werden. Diesen fehlt es jedoch meistens an Professionalität und filmischer Optik. Sie werden als fertige Filme erzeugt und sind nicht editierbar.- Des Weiteren gibt es vor allem im Spiele-Genre der Abenteuerspiele Werke, die Geschichten fast schon filmisch erzählen. Der Spieler hat nur die Möglichkeit, Einfluss auf bestimmte Elemente der Handlung zu nehmen. Ein Beispiel hierfür wäre das Spiel „Fahrenheit“, entwickelt von Quantic Dream.[6] Schon seit Jahren beigeistern Machinimas die Internet-Community. Doch diesen Filmen mangelt es häufig an filmischen Mitteln und Ausdrucksmöglichkeiten, da diese durch die voreingestellte Welt des Spiels stark limitiert sind. Sowohl Machinimas, als auch filmische Adventures haben zwei große Mankos: sie sind im Medium Videospiel gefangen und werden durch die vorgegebene Spielewelt eingeschränkt. Eine kreative Entwicklung einer filmischen Dramaturgie und filmischer Mittel ist also nicht oder nur bedingt möglich. Als Werkzeug für professionelle Filmerstellung sind sie ungeeignet.

Im Bereich des professionellen 3D-Animationsfilms ist die Konkurrenz hart und die Produktionen sind aufwändig und langwierig. Doch die Ergebnisse sind meist faszinierend. Bisher ist es kaum möglich, einen Mittelweg zwischen Machinima und konventionellem 3D-Animationsfilm zu finden. Es fehlt die Möglichkeit, schnell und effizient Filme zu erstellen, die dabei eine professionelle Qualität haben und filmische Mittel in vollem Umfang verwirklichen können. Diese Lücke zwischen professionellem 3D-Animationsfilm und Machinima kann eine Games Engine schließen, da basierend auf dieser grundsätzlich die Möglichkeit besteht, professionell und effizient zu produzieren.

5 Vgl. Basics Animation: Digital Animation, Seite 147

6 Vgl. http://www.quanticdream.com/en/game/fahrenheit-indigo-prophecy (Entnahmedatum: 02.01.2012)

Diese Bachelorarbeit richtet sich an Filmschaffende, Game-Designer, -Entwickler und 3D-Künstler. Es werden Kenntnisse im Bereich der 3D-Grafik und Grundkenntnisse iim Umgang mit dem Unreal Development Kit vorausgesetzt. Die Arbeit versucht Grenzen und Möglichkeiten von digitalen Filmen aufzuzeigen, welche mit Games Engines erzeugt werden. Sie soll außerdem einen Überblick über ästhetische und stilistische Mittel der Kinematografie geben und kann als Machbarkeitsstudie angesehen werden. Da es zum Unreal Development Kit nur wenige Quellen gibt, stützt sich diese Arbeit vor allem auf das Unreal Developmer Network[7] und Handbücher.

Hauptziel dieser Arbeit ist es zu untersuchen, ob ein professioneller 3D-Animationsfilm mit einer Games Engine erstellt werden kann, der sowohl optisch ansprechend wirkt, als auch vielfältige Gestaltungsmöglichkeiten bietet. Für die Durchführung der Machbarkeitsstudie wurde von den Autoren der speziell für Zwecke dieser Arbeit produzierte 3D-Film „Gestrandet“ erstellt, in dem diverse Aspekte von cineastischer Kreation und Filmästhetik untersucht und analysiert werden. Darauf basierend werden konkrete Vorschläge für die optimale Nutzung von Games Engines im Bereich des Filmemachens unterbreitet. Hierbei wird von der Content-Generierung bis zur finalen Ausgabe des Films jeder Arbeitsschritt betrachtet und auf seine Umsetzbarkeit in der Engine analysiert. Für die Betrachtung in der Bachelorarbeit wurde die Unreal Engine 3[8] gewählt, da sie mit Unreal Matinee, einem Werkzeug für die Erstellung von kinematografischen Sequenzen, potentiell gute Möglichkeiten für die Umsetzung von Animationssequenzen mit filmischem Anklang liefert. Die Möglichkeit, Animationsfilme mit Echtzeit-Renderern umzusetzen, wie sie in Games Engines angewandt werden, kann Zeit und Budget sparen. Neue Erzählformen können aufgegriffen werden und umgesetzt werden. Dabei geht es nicht darum, den konventionellen 3D-Animationsfilm zu ersetzen, sondern ressourcenschonend sowohl optisch als auch ästhetisch anspruchsvoll animierte Filme zu erstellen. Die Technologien des konventionellen 3D-Animationsfilms zielen darauf ab, immer realistischere Qualität zu erzeugen. Arbeitsabläufe und Technologie variieren stark gegenüber der Produktion konventioneller computergenerierter Animationsfilme, wie zum Beispiel bei Disney Pixar und anderen Studios. Diese Arbeit beschäftigt sich mit den theoretischen Grundlagen für Konzeption, Technologie und Design eines Animationsfilms mittels einer Games Engine. Unter anderem mit dieser Problematik beschaftigt sich auch das Forschungsprojekt Gamecast an der Hochschule Mittweida. Das Projekt erforscht ein Animationssystem, welches aus einer Videospielhandlung eine 3D-Animierte TV-Serie generieren kann. Im Rahmen dieses Forschungsprojekts

7 Www.udn.epicgames.com (Entnahmedatum: 02.01.2012)

8 http://www.unrealengine.com/ (Entnahmedatum: 03.01.2012)

entsteht ein System welches Emotionsdaten und Spieleraktionen aufzeichnen und weiterverarbeiten kann. Die vorliegende Arbeit nutzt das Gamecast-System um Animationsdaten für die Erstellung des Films aufzuzeichnen. Im Einzelnen wird auf konkrete Punkte eingegangen, die den Ablauf einer 3D-Filmproduktion bilden:

- Konzeption
- Erstellung der Szenen-Objekte und Charaktere
- Animation der Charaktere
- Komposition der einzelnen Elemente
- Rendering
- Postproduktion

Es wird untersucht, was bei der Konzeption zu beachten ist, um einen reibungslosen, schnellen und professionellen Arbeitsablauf für die Produktion zu erhalten. Auch auf konzeptionelle Möglichkeiten des interaktiven Entertainments wird eingegangen. Außerdem werden entstehende Probleme aufgezeigt, die bei der Erstellung von Szenen-Objekten und Charakteren für die Engine auftreten. Hierbei spielt der niedrige Detailgrad eine übergeordnete Rolle. Des Weiteren werden Möglichkeiten diskutiert, fehlende Details mit Texturen optisch zu verhüllen. Zudem wird darauf eingegangen, wie Animationen erstellt und verarbeitet werden müssen, um in der Unreal Engine eine optisch ansprechende Qualität zu erreichen. Parallel wird auch auf die Physiksimulation und ihre Vor- und Nachteile eingegangen. Zusätzlich wird in dieser Arbeit die Weiterverarbeitung der Szenen-Objekte und Charaktere, die Erstellung der Umgebung und die Einschränkungen der Echtzeitdarstellung betrachtet. Es wird untersucht, wie das Film-Projekt in Matinee organisiert werden muss, um einen effizienten Workflow zu realisieren. Hierbei werden ebenso Schwächen von Matinee aufgezeigt, die den Produktionsfluss verlangsamen. Anschließend wird auf die Postproduktion (Postprozesse, das Rendering, Schnitt des Materials) eingegangen und das Ergebnis sowie die Qualität mit konventionellen Filmen verglichen.

Die aufgeführten Aspekte werden durch die praktische Anwendung im von den Autoren entwickelten Kurzfilm „Gestrandet“ veranschaulicht.

2 Allgemeines

2.1 Überblick über Games Engines

2.1.1 Games Engines als Entwicklungsplattform

Eine Games Engine ist die Plattform, auf der ein Videospiel realisiert und dargestellt wird. Einerseits dient die Engine dem Spiel als Entwicklungsplattform, andererseits als Laufzeitumgebung.[9]

Beispiele für Games Engines sind:

- Unreal Engine (entwickelt von Epic, z.B. verwendet für die Unreal Tournament-Reihe)[10]
- CryENGINE (entwickelt von Crytek, z.B. verwendet für Crysis)[11]
- Unity (entwickelt von Unity Technologies, z.B.verwendet für Battlestar Galactica Online)[12]
- Source (entwickelt von Valve, z.B. verwendet für Half Life)[13]

Die Unreal Engine bietet eine offen verfügbare Entwicklungsplattform, genannt Unreal Development Kit (UDK), für die das Gamecast-System entwickelt wird.

Außerdem bietet das UDK mit Unreal Matinee (nachfolgend Matinee genannt) ein komfortables Tool, um filmische Sequenzen zu erzeugen. Die Kombination aus beidem unterscheidet das UDK von anderen Entwicklungsplattformen, da somit die Arbeitsabläufe für filmische Gestaltung bereichert werden.

9 Vgl. www.udk.com (Entnahmedatum: 02.01.2012)

10 http://www.unrealengine.com/ (Entnahmedatum: 02.01.2012)

11 http://www.crytec.com (Entnahmedatum: 02.01.2012)

12 http://unity3d.com/ (Entnahmedatum: 02.01.2012)

13 http://source.valvesoftware.com/ (Entnahmedatum: 02.01.2012)

2.1.2 Unreal Engine 3

Die Unreal Engine 3, die in dieser Bachelorarbeit vornehmlich betrachtet wird, erschien im Jahr 2006 und wurde von der Firma Epic Games entwickelt. Seitdem wird die Engine kontinuierlich weiterentwickelt.[14] Die Unreal Engine 3 eignet sich gut für die Erstellung von Filmen, da sie mit Unreal Matinee über einen sehr gut ausgestatten Editor für die Erstellung von Filmelementen verfügt.

Die Unreal Engine 3 besteht aus folgenden Hauptkomponenten:

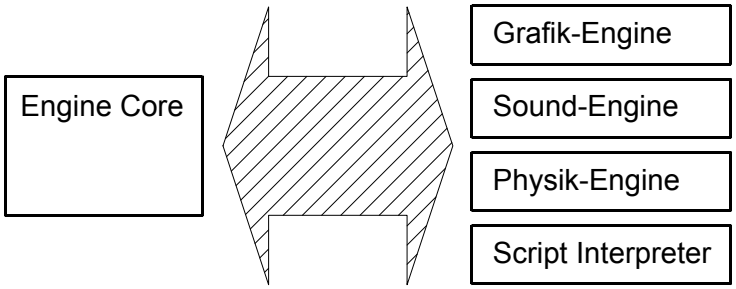

Abbildung 1: Interaktion der Engine-Komponenten[15]

Abbildung 1 zeigt die Vernetzung der verschiedenen Engine-Komponenten mit dem Rechenkern der Engine, dem Engine Core. Dieser führt alle Komponenten zusammen. Mit der Grafik-Engine wird die Darstellung des Spiels realisiert. Die Sound-Engine dient zum Abspielen von Sound-Effekten und Musik und die Physik-Engine lässt physikalische Berechnungen zu, wie zum Beispiel die Zerstörung einer Wand oder das Fallen eines Gegenstands. Der Script Interpreter dient als Schnittstelle und Übersetzer für Unreal Script, das mit diesem Element ins Spiel eingebracht werden kann. Alle diese Engines werden mit dem Engine Core zur Unreal-Engine zusammengefasst und stellen ein Spiel dar. Diese Komponenten werden im Kern der Engine, also im Engine Core zusammengeführt und damit zur Laufzeitumgebung für Spiele.

Interaktivität wird in der Engine über Kismet realisiert. Hierbei handelt es sich um einen Node-basierten[16] Script Editor. Es lassen sich Ereignisse mit Reaktionen koppeln und so die Interaktivität und Kontinuität erzeugen. Drückt man also im Spiel einen Knopf (Trigger), der im Kismet mit einem Event verknüpft wird, wird dieses ausgeführt. Mit diesem Feature der Engine wird also die Interaktivität realisiert.

---

14 Vgl. Mastering Unreal Technology, Volume 1 , Seite 11

15 Vgl. Mastering Unreal Technology, Volume 1, Seite 20

16 Node-System: Benutzeroberfläche, bei der einzelne Elemente miteinander verknüpft werden

Mit dem Werkzeug Matinee lassen sich filmische Sequenzen im Spiel erzeugen, die in Echtzeit berechnet werden. Die Matinee wird über Kismet angesteuert und kontrolliert. Der Spieler kann hierbei für gewöhnlich nicht eingreifen. Jedoch lassen sich auch Matinees erstellen, bei denen der Spieler trotzdem agieren kann, so zum Beispiel die Bewegung eines fahrenden Fahrstuhls. Der Spieler löst einen Trigger aus und die Matinee wird abgespielt. Mit Matinee lassen sich auch Sequenzen rendern. Hierbei wird direkt aus dem Editor wahlweise eine Bitmap-Sequenz oder ein unkomprimiertes Video im .avi-Format erzeugt. Jedoch sind hier bestimmte Parameter eingeschränkt, wie zum Beispiel der Umgang mit Texturen, welche die Qualität des Bildes beeinflussen. Mit verschiedenen weiteren Editoren lässt sich die Level- und Objektgestaltung umsetzen. Mit Skripten, die in der UDK-eigenen Sprache Unreal-Skript geschrieben sein müssen, lässt sich der Funktionsumfang der Engine erweitern und Abläufe automatisieren.

Da die Engine durch Epic kontinuierlich entwickelt wird, erscheinen regelmäßig Updates und neue Versionen des UDK.[17]

2.2 Konventionelle 3D-Animationsworkflows

3D-Animation bezeichnet den Prozess der Bildgenerierung und sollte nicht mit Stereoskopie[18] verwechselt werden. Der Content wird mit 3D-Modeling und computergeneriert erstellt. Oft wird Stereoskopie fälschlicherweise als 3D bezeichnet. Konventionelle 3D-Animationsfilme bedürfen einiger Konzeption, sowohl im Bereich der Projektorganisation, als auch im kreativen Schaffensprozess. In der Projektorganisation ist es nötig zu organisieren, wieviel Personal benötigt wird, um den Film zu realisieren, welcher Zeitraum eingeplant werden muss und welche Hard- und Software benötigt wird. Die Konzeption des Content, also des kreativen Teils, ist nötig, um die Projektorganisation zu definieren. Es wird also eine Story mit Drehbuch benötigt. Um den Film previsualisieren zu können braucht man ein Storyboard, welches die Kamera-Einstellung und Handlung von jeder Szene in geeigneter Form entweder als Illustration oder Animatic[19] darstellt. Charaktere müssen konzeptioniert und mit diversen Eigenschaften ausgestattet werden. Außerdem muss die Umgebung gestaltet werden Hierfür werden Konzept-Skizzen von allen relevanten Elementen erstellt. Diese dienen den 3D-Artists als Vorlage und

17 Vgl. www.udk.com/news (Entnahmedatum: 02.01.2012)

18 Stereoskopie: Zweidimensionale Bilder, die einen räumlichen Eindruck vermitteln

19 Animatic: gefilmtes Storyboard

definieren den Look des späteren Films.[20] Alle Gegenstände, Umgebungen und Charaktere müssen erstellt, mit Materialien versehen und fertig für die Animation gemacht werden. Dies ist je nach optischem Stil und Realitätsgrad sehr aufwändig und erfordert viel Aufwand. Jede Bewegung eines jeden Charakters muss animiert werden. Doch nicht nur Charaktere bewegen sich, auch die Umgebung muss dynamisch wirken und mit physikalischen Eigenschaften ausgestattet werden. So bewegen sich beispielsweise Bäume im Wind, Wolken ziehen und die Umgebung reagiert auf die Aktionen der Charaktere. Wenn das Modeling und die Animation abgeschlossen sind, müssen virtuelle Kameras gesetzt werden, die den Bildaufbau definieren.

Für eine filmische Produktion ist Licht sehr entscheidend. Mit der Beleuchtung werden Stimmungen geschaffen, Effekte erzielt und ein bestimmter Look erzeugt. Wenn all diese Schritte abgeschlossen sind, muss gerendert werden. Für jedes Einzelbild des Films wird die Ausbreitung des Lichts physikalisch berechnet und das Bild so erzeugt. Es dauert je nach Renderer und physikalischer Korrektheit einige Sekunden bis zu mehreren Minuten oder gar Stunden, bis ein Frame erstellt ist. Danach bedarf der Film einer Postproduktion, also einer Nachbearbeitung. Diese besteht meist aus Compositing[21], Farbkorrektur und Schnitt.[22]

20 Vgl. http://www.blueskystudios.com/content/process.php (Entnahmedatum: 02.01.2012)

21 Zusammenführung verschiedener Bild-Ebenen und Einsatz von Effekten

22 Vgl. http://www.blueskystudios.com/content/process.php (Entnahmedatum: 02.01.2012)

3 Konzeption von Filmen

3.1 Allgemeines

Die Konzeption von Filmen, die auf der Unreal-Engine basieren, ist im Vergleich zu konventionellen Filmen technisch eingeschränkt. Darauf wird in den folgenden Kapiteln ausführlich eingegangen. Das Unreal Development Kit bietet trotzdem genügend Möglichkeiten, um optisch und inhaltlich reizvolle Filme zu erstellen. Darunter fällt auch die Möglichkeit Interaktivität zu integrieren, die im konventionellen 3D-Animationsfilm eingeschränkter bzw. fast nicht zu realisieren ist. Wenn die Ressourcen der Engine mittels Direct-X 11[23] komplett ausgenutzt werden, kann eine recht hohe Bild-Qualität in Echtzeit erreicht werden.[24] Jedoch gibt es weitere Einsatzmöglichkeiten. Vornehmlich in Bereichen, in denen der vermittelte Inhalt relevanter als die optische Qualität ist. So lässt sich vor allem ein Einsatz in Lehrfilmen im technischen Bereich und speziell im sozial-pädagogischen Bereich vorstellen. Es können interaktive Elemente mit erzählerischen Elementen verknüpft werden und so der Lerneffekt durch Erlebbarkeit vergrößert werden. Starke Einschränkungen gibt es im Detailreichtum der Szenen und Objekte: Alle Modelle müssen Low-Poly-Objekte[25] sein. Komplexe Interaktionen zwischen den Charakteren lassen sich schwierig realisieren. Charaktere können nur bedingt auf ihre Umgebung reagieren. Beispielsweise wird eine Hand, die etwas greift, das Objekt nicht automatisch perfekt umschließen, so wie es eine reale menschliche Hand kann. Komplexere Handlungen wie dynamisches Klettern, bei dem realistisch auf die Umgebung reagiert werden muss, lassen sich also nur schwer und mit Programmieraufwand umsetzen. Dialoge und einfache Erzählelemente, wie zum Beispiel Laufen können sehr gut umgesetzt werden können.

Allgemein ist es günstiger, im fiktionalen Bereich auf fotorealistische Settings zu verzichten. Dies erleichtert die Umsetzung der Szenerie, die Erstellung der Szenenobjekte und die Charakter-Erstellung ungemein. Aus diversen Gründen, wie der Beleuchtung und der niedrigen Polygonanzahl der Charaktere, wird die fotorealistische Darstellung einer Szene mit Menschen schwer erreichbar sein. Comichafte, abstrakte und verzerrte

23 DirectX 11: Hardwarebeschleunigung für Grafik

24 Vgl. Unreal Engine 3: Samaritan Demo

25 Low-Poly: Niedrige Anzahl von Polygonen (griechisch: Vieleck, in der 3D-Computergrafik werden Objekte aus Polygonnetzen modelliert)

Formen dienen bewusst dazu, Phantasiewelten zu erstellen und nicht Vorhandenes abzubilden. Auch ist es möglich, mit filmischen Mitteln Defizite zu kaschieren. So lässt sich zum Beispiel mit Schärfentiefe ein aufwändiger Hintergrund vermeiden. Bewegungsunschärfe sorgt nicht nur für Lebendigkeit im Bild, sondern kann auch fehlende Details verbergen. Die Official Samaritan Demo von Epic demonstriert eindrucksvoll, wo die Möglichkeiten der Engine-Technologie liegen.[26] Die Demo zeigt, dass es möglich ist, mit leistungsstarken Rechnern interessante Settings zu erstellen, die sich anspruchsvoller Effekte bedienen und filmische Mittel realisierbar machen.

3.2 Umgebungs-Design

Das Design der Umgebung ist dank den Werkzeugen des UDK sehr vielfältig. So ist es möglich, mit Hilfe des Terrain-Editors schroffe Berglandschaften zu erstellen, sowie auch Strand und weiche Hügel, auch urbane Settings zu erstellen stellt kein Problem dar. Diese können je nach Rechenleistung des vorhandenen Computers auch sehr detailreich sein. Dadurch ergeben sich viele Möglichkeiten Umgebungen zu schaffen, die anspruchsvoll wirken, aber das Echtzeit-Rendering optimal funktionieren lassen.

3.3 Objekt- und Charakter-Design

Das Design der Assets ist eingeschränkter. Durch Begrenzung der möglichen Rechenleistung beim Echtzeit-Rendern müssen Polygone gespart werden. Objekte sollten nicht zu detailüberladen sein. Aus diesen Gründen ist es notwendig, 3D-Objekte mit einer geringen Anzahl an Polygonen zu erstellen. Durch effizientes Modeling der Charaktere und Szenenobjekte lassen sich so Ressourcen sparen, die dann für andere Objekte innerhalb der Szenerie zur Verfügung stehen. Vor allem für den Betrachter unwichtige Randobjekte sollten in ihrer Polygonanzahl stark begrenzt sein. Hierdurch werden mehr Ressourcen für wichtige Objekte frei.

Wie auch im konventionellen 3D-Animationsfilm üblich, bietet es sich an, Konzeptzeichnungen zu erstellen, um Objekte und Charaktere zu previsualisieren. Dadurch lassen sich Details überlegter und nur wenn nötig einbringen. Die strikten Polygon-Einsparungen erfordern im Voraus eine genaue Planung der Bilder. Es sollte demzufolge immer ein Storyboard für die Visualisierung des Films erstellt werden, welches genau verdeutlicht, welche Objekte in welcher Einstellung zu sehen sind. Daraus lässt sich

26 http://www.youtube.com/watch?v=XgS67BwPfFY&hd=1 (Entnahmedatum: 02.01.2012)

dann eine Objekt-Liste ableiten, die dann in die jeweiligen Detailgrade aufgeteilt wird. So bleibt der Arbeitsablauf organisiert und das Projekt wird sowohl für Auftraggeber, als auch für die Beteiligten visualisiert.

3.4 Konzeption der Postproduktion

Durch das Echtzeit-Rendering gibt es mehrere Möglichkeiten, um die Produktion bis zum Ende hin variabel zu halten. So kann jede Kamera über die ganze Szene gerendert werden und dann in der Postproduktion zum Beispiel per Multi-Cam-Schnitt entschieden werden, wie der Film geschnitten sein soll. Eine andere Möglichkeit wäre es, den Schnitt schon vor dem eigentlichen Rendering vorzunehmen und mit einer DirectorGroup im Matinee, also einer Art Bildmischer, Kameras anzusprechen.

Mit dem Echtzeit-Rendering besteht immer die Möglichkeit, das Endresultat zu überprüfen. Das UDK erlaubt es bereits im Programm Szenen zu montieren. So lässt sich ein Eindruck vermitteln, wie der Film letztendlich fertig aussehen soll. Auch ist es möglich, erstellte Szenerien mittels Exporter in 3D-Bearbeitungsprogrammen weiterzuverarbeiten. So lassen sich anspruchsvolle und physikalisch korrektere Renderings erstellen. Jedoch sollte in der Konzeption bedacht werden, dass dadurch wesentlich höhere Renderzeiten und ein Mehraufwand für die weitere Bearbeitung entstehen.

3.5 Interaktive Elemente

Durch die Verwendung einer Spiele-Entwicklungsumgebung liegt es nahe, auch interaktive Elemente einzubringen. Diese lassen sich im UDK durch Kismet einfügen. Kismet ist ein Skriptwerkzeug, mit dem sich beispielsweise Interaktionen oder Events durch bestimmte Voraussetzungen auslösen lassen. So lassen sich interaktive Elemente einbringen.

4 Erstellung von Charakteren und Objekten

4.1 Modeling für Games Engines

4.1.1 Allgemeines

Durch die Verwendung einer Games Engine zur Entwicklung eines digitalen Films ergeben sich für die Erstellung der Charaktere und Szenenobjekte Einschränkungen. Die Games Engine limitiert die maximale Anzahl an Polygonen, aus denen ein 3D Modell bestehen kann. Bei der verwendeten Unreal Engine 3 beträgt dieses Limit beispielsweise 32.000.[27] Zudem wird die Polygon-Anzahl der Szene auf ca. 2 Millionen Polygone beschränkt. Dieser Wert ergibt sich aus allen in der Szene befindlichen Objekten.

Bei der Verwendung von 3D-Objekten in einer Games Engine ist zu beachten, dass alle Polygone in Dreiecke umgewandelt werden. Dies ergibt sich daraus, dass handelsübliche Grafikkarten nur Dreiecke verarbeiten können. Beim Modeling der Objekte ist somit primär die finale Anzahl der Dreiecke relevant. Jedes vierseitige Polygon wird beim Import in die Engine in zwei Dreiecke umgewandelt. Dreiecke werden weiterhin als solche behandelt. N-Ecke werden in die Anzahl von Dreiecken aufgeteilt, welche nötig ist, um alle Eckpunkte der Geometrie abzudecken. Da diese automatische Aufspaltung unter Umständen zu fehlerhafter Geometrie führen kann, ist es empfehlenswert, die Polygone bereits vorher soweit wie möglich in Drei- und Vierecke aufzuteilen. Dadurch lässt sich das Endergebnis besser kontrollieren. Diese Aufteilung kann manuell vor dem Export geschehen. Bei der Verwendung des Autodesk-FBX-Formates lässt sich diese Triangulation auch über den Exporter durchführen. Wurde das Objekt bis zum Export in das FBX-Format nicht bereits vollständig zu Dreiecken konvertiert, erfolgt die automatische Umwandlung beim Import in das UDK.[28]

Objekte sollten aus einem Stück modelliert werden. Das heißt, ineinander ragende Objekte sollten soweit wie möglich vermieden werden.[29] Dies kann später für Shadingprobleme innerhalb der Engine sorgen. Shading bezeichnet die Schattierung von Objekten

27 http://udn.epicgames.com/Three/ImportingMeshesTutorial.html (Entnahmedatum: 02.01.2012)

28 http://udn.epicgames.com/Three/FBXStaticMeshPipeline.html#Triangulation (Entnahmedatum: 02.01.2012)

29 http://udn.epicgames.com/Three/LightMapUnwrapping.html (Entnahmedatum: 02.01.2012)

innerhalb der Computergrafik. Hierbei handelt es sich um ein Interpolationsverfahren, bei dem der Normalenvektor eines Polygons verschoben wird um beispielsweise Oberflächen glatter aussehen zu lassen, dies sind beispielsweise das Phong- und Blinn-Shading[30]. In den Testszenen ließ sich feststellen, dass durch fehlerhaft erstellte Geometrie das Shading negativ beeinflusst wird. Es entstehen, wie in Abbildung 2 ersichtlich wird, Schattenausblutungen, die das Erscheinungsbild der Szene beeinflussen. Dies wird vor allem bei großen statischen Objekten deutlich.

Abbildung 2: Shadingprobleme bei nicht zusammenhängender Geometrie (Eigene Darstellung)

Das UDK akzeptiert auch den Import von Vertex-Colors. Vertex-Colors definieren Farbwerte für die Eckpunkte von 3D Modellen, dies zeigt Abbildung 3. Diese Farben lassen sich im UDK-Materialeditor ansprechen. Über diesen lassen sich beispielsweise Farbwerte von Objekten ändern oder Überblendungen mehrerer Texturen schaffen. Mittels einer speziellen Wind-Funktion lassen sich auch die Auswirkungen von Wind auf Objekte simulieren. Die Stärke der verwendeten Farbe definiert hierbei die Stärke der Auswirkung der externen Kraft. Bei Pflanzen können zur Animation der Windbewegung beispielsweise die Blätter mit einer roten Farbe definiert werden, während die Äste nur eine leichte bis keine Färbung besitzen. Dies sorgt bei der späteren Windsimulation im UDK dafür, dass die Blätter vom Wind beeinflusst werden, während die Äste sich nicht oder kaum bewegen. Die Auswahl der jeweiligen Vertex-Color erfolgt im späteren Verlauf der Weiterverarbeitung im Materialeditor des UDK.

30 Vgl. Mental Ray for Maya, 3ds Max and XSI, Seite 228

Abbildung 3: Vertex-Colors bei einem Ast (Eigene Darstellung)

Die verwendete Unreal Engine 3 nutzt ein generisches Einheitensystem, diese Einheiten heißen Unreal Units (UU). Eine UU wird entweder zu zwei Zentimetern oder einem Zoll umgerechnet. Bei der Charakter- und Objekterstellung muss deshalb vorher darauf geachtet werden, alle Modelle in diesem Maßstab zu generieren. Beispielsweise ist ein Charakter, der 180 cm groß ist, im UDK 95 UU hoch.

4.1.2 Modeling

In konventionellen 3D-Animationsfilmen spielt die Anzahl der Polygone einer Szene eine untergeordnete Rolle. Die Szenen müssen nicht in Echtzeit grendert werden und meist kann eine enorme technische Ausstattung genutzt werden. Hierdurch kann eine Szene aus tausenden detailreichen Modellen aufgebaut werden, die wiederum aus mehreren tausend bis hin zu Millionen Polygonen bestehen. Die durch die implementierten Features und Leistung der Games Engine verursachten Einschränkungen in der Anzahl der maximalen Polygone resultieren in einem zwangsweise verringerten Detailgrad der Objekte. Um dennoch einen akzeptablen Detailgrad gewährleisten zu können, werden so genannte Normal Maps (deutsch etwa: Normalenrelief) verwendet.

Abbildung 4: Objekt mit (links) und ohne Normal Map (rechts) (Eigene Darstellung)

Durch Anwendung von Normal Maps können weiche Übergänge an Kanten und zusätzliche Details auf Objekten simuliert werden ohne die Geometrie zu verändern[31]. Dies wird in Abbildung 4 verdeutlicht. Diese weichen Übergänge an Kanten sind notwendig, da selbst hart wirkende Objekte in der realen Welt über leichte Abrundungen verfügen. Erkennbar ist dies beispielsweise bei einer Tischkante. Mit diesen detailarm modellierten Objekten und Normal Maps lassen sich Szenen erstellen, die aus einer wesentlich geringeren Anzahl an Polygonen bestehen. Dies ist vor allem bei Echtzeitdarstellungen wichtig, kann aber auch für das konventionelle Rendern von Szenen und Animationen genutzt werden.[32] Durch das effektive Einsetzen von Normal Maps lassen sich Szenen mit einem hohen Detailreichtum erzeugen, die dennoch ressourcensparend sind.

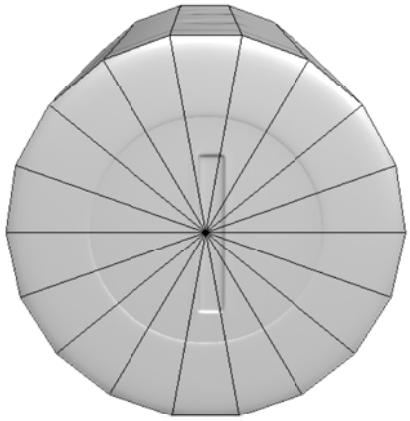

Abbildung 5: Normal Map unter flachem Betrachtungswinkel (Eigene Darstellung)

Eine Normal Map besteht aus drei Farbkanälen mit jeweils acht Bit Farbtiefe. Sie besitzt somit mindestens 24 Bit Farbtiefe. Durch die Nutzung eines Alphakanales kann sie auf 32 Bit erweitert werden. Dies wirkt sich allerdings nicht auf die Normal Map selbst aus, sondern ermöglicht es lediglich, den Kanal für eine Maske zu nutzen. Die Farbkanäle der Normal Map stehen hierbei für verschiedene Achsen des kartesischen Koordinatensystems. Rot steht für die Ausrichtung in x-Richtung, während Grün die Ausrichtung von in y-Richtung wiedergibt. Der blaue Farbkanal stellt die vertikale Tiefe dar. Der Grundfarbton einer Normal Map hat den RGB-Wert von (128,128,255). Dieser Blauton steht für eine senkrechte Oberflächennormale, welche die Lichtberechnung auf der Geometrie nicht beeinflusst. Abweichungen dieser Farbe sorgen für optische Erhöhungen und Vertiefungen auf dem Modell. Für die Anwendung im UDK werden nur

31 Vgl. Rendering & Lightning, Seite 297

32 Vgl. Autodesk 3ds Max 2011 Help, Seite 7320

Normal Maps akzeptiert, deren grüner Farbkanal bei der Erstellung nach unten ausgerichtet ist.[33] Ein nach oben zeigender Grünkanal sorgt im UDK ansonsten für eine fehlerhafte Darstellung der Normal Map. Die Schattierung des Objektes erscheint dann gespiegelt. Anhand der in der Normal Map enthaltenen Farbinformationen lassen sich auf flachen Objekten Oberflächendetails simulieren, die hervorstehen oder eingestanzt sind. Erst aus flachen Betrachtungswinkeln wird ersichtlich, dass es sich lediglich um flache Geometrie handelt. Abbildung 5 zeigt ein Objekt mit Normal Map aus einem flachen Betrachtungswinkel. Die Testszenen zeigen also, dass es möglich ist, mit Normal Maps im Modell fehlende Details zu kompensieren. Um eine Normal Map aus einem Highpoly-Modell zu generieren, muss dieses entweder durch die Verwendung von Subdivision Modeling oder durch Sculpting erstellt werden. Highpoly-Modelle besitzen eine sehr hohe Anzahl an Polygonen. Sie werden in konventionellen 3D-Animationsfilmen verwendet. Für die Verarbeitung in Games Engines ist deren Detailgrad zu hoch. Subdivision Modeling (deutsch etwa: Unterteilungs-Modellierung) erfolgt in gängigen Programmen mittels spezieller Modifikatoren, während beim Sculpting (deutsch: Bildhauerei) auf Programme zurückgegriffen wird, die es ermöglichen, 3D-Modelle wie Tonskulpturen zu formen. Beispiele für Sculpting-Werkzeuge sind Autodesk Mudbox[34] oder Pixologic ZBrush[35] sowie die freie ZBrush-Variante Sculptris.[36]

Um während der Contentgenerierung bereits das finale 3D-Modell inklusive Normal Map betrachten zu können, empfiehlt sich die Verwendung eines Direct-X Shaders.[37] Mit diesem ist beispielsweise bei 3ds Max[38] eine Vorschau im Echtzeitviewport möglich. Ein solcher Shader ist der 3Point Shader Lite von 3Point Studios.[39]

33 http://udn.epicgames.com/Three/CreatingNormal Map.html#Normal Map Formats (Entnahmedatum: 02.01.2012)

34 http://usa.autodesk.com/adsk/servlet/pc/index?id=13565063&siteID=123112 (Entnahmedatum: 06.01.2012)

35 http://www.pixologic.com/zbrush/ (Entnahmedatum: 06.01.2012)

36 http://www.pixologic.com/sculptris/ (Entnahmedatum: 02.01.2012)

37 Vgl. Autodesk 3ds Max 2011 Help, Seite 7320

38 Vgl. http://www.autodesk.de/adsk/servlet/pc/index?id=14642267&siteID=403786 (Entnahmedatum: 02.01.12)

39 http://www.3pointstudios.com/3pointshader_about.shtml (Entnahmedatum: 02.01.2012)

4.1.3 Projektionsmapping

Die erzeugten detailreichen Modelle müssen zur Verwendung innerhalb der Games Engine auf ihre jeweilige undetaillierte Version projiziert werden. Gängige 3D-Bearbeitungsprogramme besitzen hierfür ein Dialogfenster, mit dem sich die Projektion realisieren lässt. Dieses undetaillierte Objekt benötigt im Gegensatz zum Highpoly-Objekt UVW-Koordinaten.[40]

Als mögliche Verfahren zur Erzeugung der Normal Map wäre hierbei das Raycastingverfahren[41] (deutsch etwa: Strahlenwurf) oder die Anwendung eines Projektionskäfigs zu nennen. Beim in Abbildung 6 gezeigten Raycastingverfahren wird von der Oberflächennormale eine Abtastung nach innen und außen vorgenommen. Durch diese Abtastung kann ein Höhenunterschied registriert werden. Dieser Höhenunterschied wird in die Normal Map geschrieben. Problematisch bei diesem Verfahren sind harte 90°-Kanten, da aufgrund der geradlinigen Abtastung somit Lücken beziehungsweise doppelte Abtastungen entstehen können.

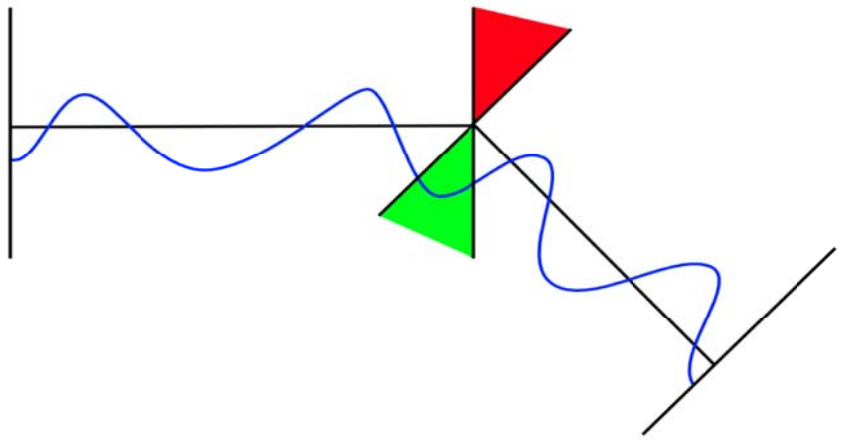

Abbildung 6: Raycasting: doppelte Abtastung grün und fehlende rot. Highpoly Geometrie blau (Eigene Darstellung)

Das Projektionskäfigverfahren arbeitet mit einem zweiten 3D-Modell, welches als Cage bezeichnet wird. Dieses Modell umgibt das Highpoly-Modell des 3D-Modells. Bei diesem Verfahren wird nur von innen nach außen abgetastet. Geometrie, die sich also außerhalb des Projektionskäfigs befindet, wird abgeschnitten und erscheint als fehlerhafter Bereich in der Normal Map. Dies resultiert in schwarzen Flächen im Renderer. Um diese fehlerhaften schwarzen Flächen zu vermeiden, sollte der Käfig das Modell umschließen, wiederum aber von ihm nicht zu weit entfernt sein, da dies die Genauigkeit

40 Vgl. Autodesk 3ds Max 2011 Help, Seite 7320

41 Vgl. „Virtuelle Welten mit der Raycasting-Technik darstellen“ Seite 246

verringert. Je höher die Distanz des Käfigs zum Objekt, desto ungenauer wird die Geometrie projiziert. Ungenaue Einstellungen des Projektionskäfigs können in welligen oder schiefen Normal Maps resultieren. Dies kann beispielsweise bei einem Objekt auftreten, in dessen Normal Map eine schmale Einkerbung projiziert wird. Dies lässt sich entweder durch einen genaueren Käfig oder mittels Bildbearbeitungssoftware beheben. Um den späteren Bearbeitungsaufwand so gering wie möglich zu halten, sollte bereits bei der Projektion auf möglichst hohe Genauigkeit geachtet werden. Bei kleineren Fehlern innerhalb der Normal Map, wie beispielseise farbigen Artefakten, bietet es sich an, diese mit Bildbearbeitungssoftware zu beheben.[42] Parallel zur Normal Map lässt sich direkt eine Ambient Occlusion Map[43] erstellen. Mit dieser lassen sich im späteren Verlauf noch zusätzliche Schattierungen in die Textur eines Objektes einbringen. Die Ambient Occlusion Map ist eine 2D-Textur, welche Schattierungen unter Berücksichtigung der gegenseitigen Verdeckung (englisch: Occlusion) von Objekten berechnet. Daraus resultiert, dass sich Objekte, die nah beieinander liegen, gegenseitig schattieren. Zu beobachten ist dies auch in der realen Welt, beispielsweise in den Ecken eines Zimmers. Licht wird reflektiert. Der Grund hierfür ist, dass Licht reflektiert und absorbiert wird.

In 3D-Renderern, wie zum Beispiel Mental Ray[44], wird dieses Verhalten nachempfunden, wodurch sich Ambient Occlusion Maps generieren lassen. Photonenmapping bedeutet, dass Lichtteilchen in den Raum geschossen werden. Deren Reflexionen im Raum werden solange verfolgt, bis sie von einer diffusen Oberfläche absorbiert werden.[45] Da beispielsweise bei Zimmerecken die Aufenthaltswahrscheinlichkeit eines Lichtteilchens geringer ist als im Zentrum des Raums, werden diese automatisch dunkler dargestellt. Die oben genannten Verfahren sind allerdings sehr rechenintensiv. Aus diesem Grund wird Ambient Occlusion in Games Engines derzeit noch sehr spärlich eingesetzt. Meist wird dieser Effekt in die Texturen eingebrannt. Dies sorgt allerdings für statisch wirkende Szenen, da sich diese Ambient Occlusion nicht im Spielverlauf ändern kann. Echtzeitberechnete Ambient Occlusion beherrscht derzeit beispielsweise die Unreal Engine 3 sowie die Frostbite 2 Engine[46]. Über spezielle Treibereinstellungen

42 Vgl. Autodesk 3Ds Max 2011 Help, Seite 7320

43 Vgl. Lightning & Rendering, Seite 75

44 Siehe: Mental Ray for Maya, 3ds Max and XSI, Seite 2

45 Vgl. Autodesk 3Ds Max 2011 Help, Seite 7195 ff

46 http://www.frostbite2.com/ (Entnahmedatum: 02.01.2012)

lassen sich Effekte, wie Ambient Occlusion derzeit auch mit nVidia-Karten in Videospielen aktivieren, die diese Technik nicht unterstützen.[47]

4.2 UVW-Maps

UVW-Maps legen die Texturkoordinaten eines 3D-Objektes fest. Dies ist nötig, da definiert werden muss, wie zweidimensionale Texturen auf einem dreidimensionalen Modell angeordnet sind. Das UVW-Koordinatensystem ist ein kartesisches Koordinatensystem mit drei Achsen. Die U-Achse entspricht hierbei der X-Achse. Y und Z werden von den Koordinaten V und W definiert. Für die meisten Texturen werden nur die Koordinaten U und V benötigt. Dreidimensionale Texturen[48] sind in der verwendeten Unreal Engine beispielsweise nicht nutzbar, weshalb die Z-Achse wegfällt. Die meisten gängigen 3D-Programme bieten einen UVW-Editor zum Bearbeiten der UVW-Maps.[49]

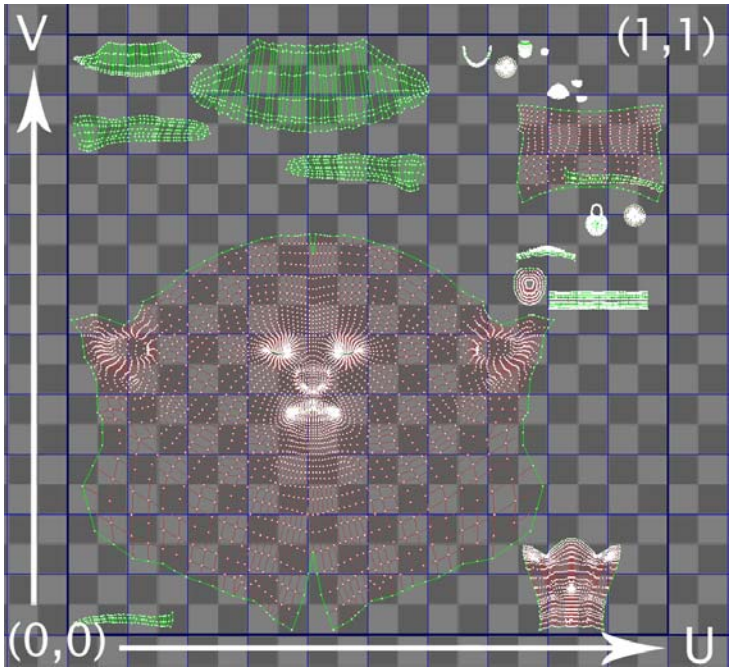

Abbildung 7: kartesisches Koordinatensystem des 3ds Max UVW Editors (Eigene Darstellung, Charakter Pixable)

47 http://www.nvidia.de/object/win7-winvista-64bit-290.36-beta-driver-de.html (Entnahmedatum: 02.01.2012)

48 Vgl. Rendering & Lightning, Seite 328

49 Vgl. Rendering & Lightning, Seite 323

4.2.1 Allgemeines

Innerhalb der UVW-Map sind die Flächen des 3D-Objektes planar abgebildet. Bei konventionellen 3D-Animationsfilmproduktionen oder 3D-Renderings reicht es, für viele Objekte bereits die UVW-Map planar von allen Seiten eines Würfels zu erzeugen, um eine Textur über ein Objekt zu spannen. Daraus resultieren überlappende UVW-Flächen. In Games Engines wird dies meist für die Verwendung von kachelbaren Texturen für urbane Szenerien genutzt. Diese Kachelung hat auch zur Folge, dass sich Szenenobjekte mittels kachelbarer Texturen versehen lassen. Diese ermöglichen es, ein Objekt detailliert und scharf wirken zu lassen, obwohl die verwendete Textur nur eine geringe Auflösung hat. Nachteil dieser Methode ist, dass es dadurch nicht möglich ist, markante Details einzubinden, da sich diese ebenfalls immer wiederholen würden. Diese Details sind je nach Entfernung zum Betrachter mehr oder weniger deutlich erkennbar.

4.2.2 Anwendung von UVW-Maps

Bei herkömmlichen Produktionen ist es irrelevant, in welchen Bereichen der UVW-Map sich die UVW-Flächen befinden. Bei Games Engines ist explizit darauf zu achten, dass sich alle Flächen im (0,0) zu (1,1) Bereich des UVW-Koordinatensystems befinden. Die Koordinate (0,0) definiert die linke untere Ecke der später verwendeten Textur. (1,1) ist die rechte obere Ecke. Dieser Bereich stellt die Textur dar und wird an seinen Grenzen gekachelt. Dieser ist in den meisten 3D-Bearbeitungsprogrammen im UVW-Editor durch ein Raster definiert. Eine Ausnahme stellen hierbei gespiegelte oder doppelt vorhandene Teile der Geometrie sowie kachelbare Objekte dar. Dies wird in Abbildung 8 veranschaulicht. Die Teile werden im UVW-Editor mit einem Offset versehen, so dass sie um den Wert eins in eine beliebige Richtung versetzt werden können. Hierdurch wird eine Kachelung der Textur möglich, sowie die Anwendung einer Normal Map auf gespiegelte oder mehrfach vorhandene Teile der Geometrie.[50]

50 Vgl. http://udn.epicgames.com/Three/MaterialBasics.html#Texture Scale / Texture Tiling (Entnahmedatum: 02.01.2012)

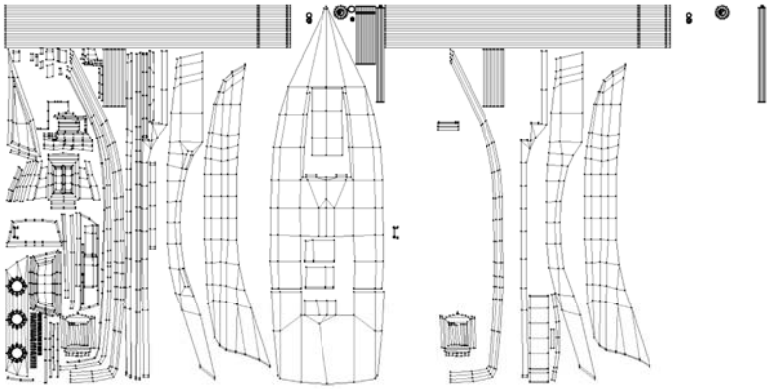

Abbildung 8: Offset gespiegelter UVW-Koordinaten (Eigene Darstellung)

4.2.3 UVW-Seams

Seams sind deutlich sichtbare Kanten im 3D-Modell, die auftreten, wenn zwei zusammenhängende Polygone sich auf getrennten UVW-Flächen befinden. Dies wird vor allem in Verbindung mit Normal- und Lightmaps deutlich sichtbar[51]. Diese stellen einen harten Bruch in der Glättung des Objektes dar und lassen sich nicht vermeiden. Es bietet sich deshalb an, die UVW-Seams an Teile des 3D-Modells zu verlagern, die nicht sofort sichtbar sind oder von anderer Geometrie verdeckt werden. Bei menschlichen Gesichtern ist es beispielsweise empfehlenswert, die Seam unter die Haare zu legen und den Rest des Gesichtes weitestgehend aus einem Stück aufzufalten.

Die Anordnung der Seams sorgt in Verbindung mit Normal Maps für ein weiteres Problem. Dies wird in Abbildung 9 deutlich. Es wurde festgestellt, dass bei nicht geradlinig ausgerichteten UVW-Flächen deutlich sichtbare Seams auftreten. Diese lassen sich beheben, indem die Geometrie exakt auf die U- und V-Achse ausgerichtet wird. Außerdem wird bei falsch angeordneten UVW-Flächen das Texturieren der Modelle im Bildbearbeitungsprogramm erschwert. Nicht zusammenhängende UVW-Flächen weisen meist deutliche, nur schwer behebbare Brüche im Texturfluss auf. Es konnte festgestellt werden, dass sich diese Fehler vermeiden lassen, indem Flächen soweit wie möglich zu einem Verbund zusammengefasst werden. Hierdurch wird die spätere Texturierung erleichtert und die Gefahr von Shading-Problemen bei der Verwendung von Normal Maps wird reduziert.

51 Vgl. http://udn.epicgames.com/Three/LightMapUnwrapping.html#On Creating Lightmaps (Entnahmedatum: 27.12.2011)

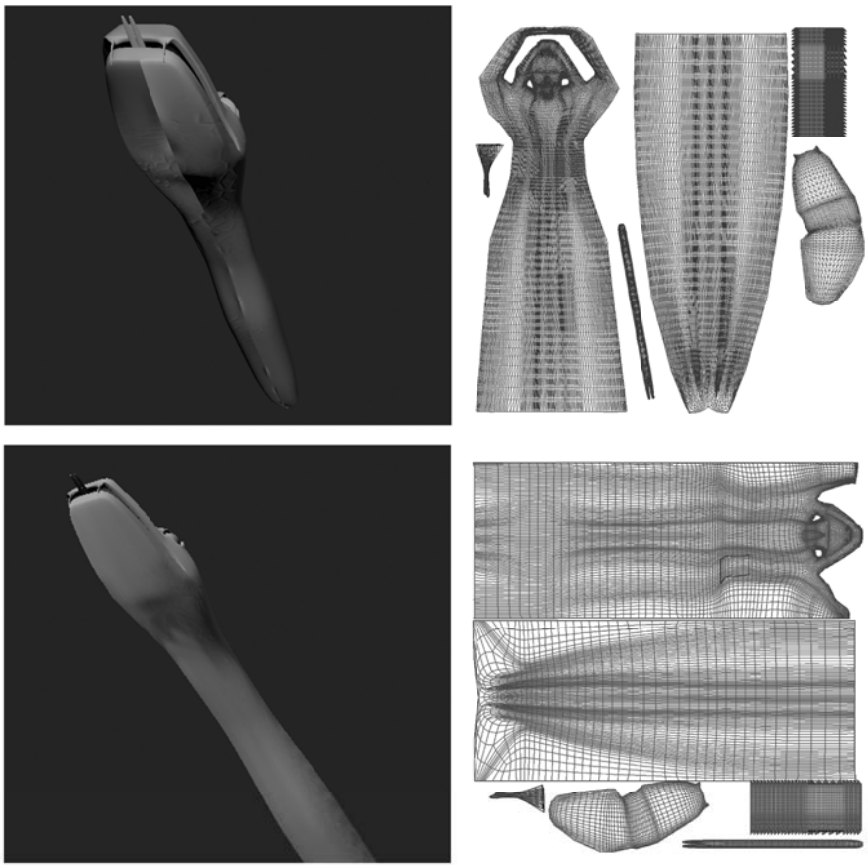

Abbildung 9: Sichtbare UVW Seam und UVW Map (oben) Korrekte UVW Seam und UVW Map (unten) (Eigene Darstellung)

Eine weitere Möglichkeit Seams zu vermeiden, ist das Einbringen von zusätzlicher Geometrie an problematischen Kanten. Dies wird in Abbildung 10 deutlich. Das Einbringen zusätzlicher Knotenpunkte hat jedoch zur Folge, dass sich vor allem bei komplexer Geometrie die Anzahl der Polygone stark erhöht. Die nachfolgende Abbildung verdeutlicht diese Problematik.

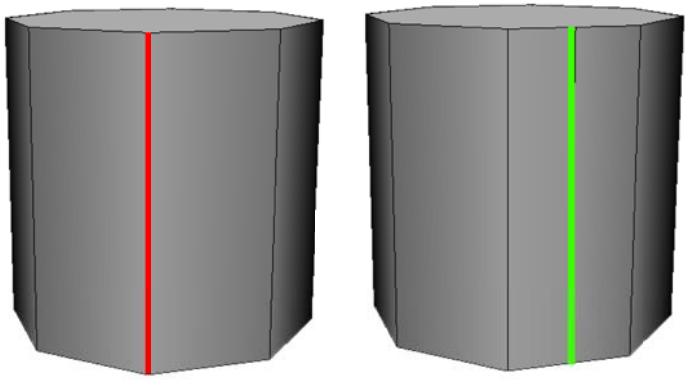

Abbildung 10: Fehlerhafte (rot) und optimale Seam (grün) (Eigene Darstellung)

4.2.4 Mipmaps

Die automatische Distanzanpassung der Texturen innerhalb der Engine bereitet Umstände. Die Engine erstellt automatisch verkleinerte Texturen für die Distanzdarstellung, sogenannte Mipmaps. Diese werden benötigt, um ein Flackern der teilweise sehr hoch aufgelösten Texturen auf Distanz zu verhindern.[52] Dieses automatische Weichzeichnen der Texturen setzt einen geeignet hohen Abstand der einzelnen UVW-Flächen zueinander voraus. Um eine korrekte Darstellung der verkleinerten Texturen zu gewährleisten, ist ein Padding (deutsch: Auffüllung) nötig. Padding bezeichnet das Auffüllen von unbenutzten Texturbereichen mit Bildinformationen. Beim Padding werden diese undefinierten Texturbereiche wie in Abbildung 11 mit Bildinformationen gefüllt. Erfolgt dies nicht, kann es zu fehlerhafter Darstellung der verkleinerten Texturen kommen. Dies wird beispielsweise bei Texturen von Blättern deutlich. Ohne Padding wird bei zunehmender Distanz zum Betrachter ein deutliches Ausbluten der Textur im Bereich des Transparenzübergangs sichtbar.[53] Durch das Füllen der nicht verwendeten Texturfläche mit einer weich gezeichneten Variante der Textur wird dieses Problem beseitigt. Das Erstellen von Padding für Texturen lässt sich automatisieren. Hierzu wird Bildbearbeitungssoftware benötigt, welche das Ausführen von Aktionen unterstützt. Dies ist beispielsweise bei Adobe Photoshop[54] möglich. Hierbei muss der für das Padding notwendige Arbeitsablauf nur einmal aufgezeichnet werden. Er besteht in der Regel aus mehreren überlagerten Ebenen, die unterschiedlich stark weichgezeichnet sind. Für alle weiteren Texturen, bei denen dies benötigt wird, lässt sich der Vorgang automatisiert. Bei der Erstellung der Testszenen erwies dies sich als effektiv

Abbildung 11: Textur ohne (links) und mit Padding (rechts) (Eigene Darstellung)

52 Vgl. http://udn.epicgames.com/Three/Landscape.html (Entnahmedatum: 02.01.2012)

53 Vgl. http://udn.epicgames.com/Three/LightMapUnwrapping.html#On Creating Lightmaps (Entnahmedatum: 02.01.2012)

54 Vgl. http://www.adobe.com/de/products/photoshop.html (Entnahmedatum: 02.01.2012)

4.2.5 Lightmap UVW

Statische, in eine Games Engine importierte Objekte werden oft mit einer festen Beleuchtung versehen. Dies sorgt in Verbindung mit UVW Mapping zu einem sichtbaren Problem, da die Beleuchtung in die Texturinformationen übertragen wird. Überlappungen der UVW-Flächen oder kachelbare Bereiche, wie zum Beispiel bei der Fassade eines Hochhauses, haben hier auch ein Kacheln der Lichtberechnung. Um dieses Problem zu beheben, wird ein zweiter UVW-Kanal für das 3D-Modell benötigt. Die meisten 3D-Programme bieten es im UVW-Editor an, die Map auf einen spezifischen Kanal zu legen. Der erste Kanal, welcher im UDK als Kanal 0 bezeichnet wird, beinhaltet die Texturinformationen. Der zweite Kanal wird für die Lightmaps der statischen Beleuchtung genutzt. Bei der Lightmap UVW ist es wichtig, dass keine Überlappungen der Geometrie existieren.[55]

Bereits bei Erstellung der Lightmap UVW muss die Texturgröße der finalen Lightmap berücksichtigt werden. Über diese wird der Abstand sowie die Anordnung der einzelnen UVW-Flächen definiert. Rendering-Tests habn ergeben, dass die Einstellung des Grids entscheidend für das spätere Erscheinungsbild der Lightmap ist. Hierfür muss ein Einrasten am Raster im UVW-Editor aktiviert sein, welches in Abhängigkeit der Auflösung der finalen Lightmap eingerichtet wird. Diese Auflösung ist abhängig von der Größe des Objektes sowie dessen Entfernung von der Kamera. Gängige Auflösungen für Lightmaps sind 32x32px oder 128x128px.[56]

4.3 Texturierung für Games Engines

Zur Anzeige von 3D-Objekten in Filmen sowie Videospielen werden Texturen benötigt, die das Aussehen der Objekte definieren. Hierbei handelt es sich in der Regel um 2D-Texturen, welche über die UVW-Map auf das 3D-Modell angewendet werden. Ausnahmen bilden hierbei 3D-Texturen, welche zusätzliche Tiefeninformationen beinhalten. Dies hat den Vorteil, dass Objekte wie beispielsweise ein Baum beim Zerschneiden eine innere Struktur aufweisen. Das Unreal Development Kit unterstützt nur zweidimensionale Texturen, weshalb auf spezielle dreidimensionale Texturen nicht eingegangen wird.

55 Vgl. http://udn.epicgames.com/Three/LightMapUnwrapping.html#Lightmap Coordinates Index (Entnahmedatum: 02.01.2012)

56 Vgl. http://udn.epicgames.com/Three/LightMapUnwrapping.html#On Creating Lightmaps (Entnahmedatum: 02.01.2012)

4.3.1 Allgemeines

Prinzipiell sind für die Arbeit im UDK drei Texturen nötig: die Diffuse-[57], Specular-[58] und Normal Map. Das UDK unterstützt Texturen der Bildformate Bitmap sowie Targa. Targa besitzt im Gegensatz zu Bitmap die Vorteile einer Komprimierung sowie die Möglichkeit, einen Alphakanal direkt in die Bilddatei zu schreiben. Dies resultiert zwar in einer höheren Dateigröße, ermöglicht es allerdings, mehr Informationen zu speichern. Das UDK unterstützt nur Texturauflösungen, die auf Zweierpotenzen aufbauen. Gängige Texturauflösungen sind beispielsweise 512x512px oder der Maximalwert von 4096x4096px.[59] Höhere Auflösungen resultieren in einem höheren Speicher- und Leistungsbedarf in der Engine. Deshalb sollten Auflösungen überlegt gewählt werden. Die Anwendung von unterschiedlichen Seitenverhältnissen ist möglich, beispielsweise 512x128px.

4.3.2 Diffuse Map

Die Diffuse Map definiert das Aussehen eines Objektes, als wäre es von allen Seiten gleichmäßig beleuchtet. In der Diffuse Map werden alle für das Aussehen relevanten Details, wie zum Beispiel Beschriftungen oder Kleidungsmuster gespeichert. Für die Arbeit in Games Engines kann zusätzlich noch die Beleuchtung einer Szene fest in die Diffuse Map eingebrannt werden. Dies ermöglicht es, präzisere statische Beleuchtungen zu erstellen, als dies mit Lightmaps möglich wäre, da hierbei die volle Auflösung der Textur benutzt werden kann. Problematisch ist es, sobald sich UVW-Flächen überlappen. Dann würden auch, wie bei einer Lightmap, die Lichtinformationen gedoppelt auftreten, weshalb sich das Einbacken von Beleuchtungsinformationen in die Diffuse Map nur für kleinere Objekte oder die Ambient Occlusion eignet.

Die Wahl der Auflösung für die Diffuse Map ist abhängig von der Größe und der Entfernung des Objektes zur Kamera. Beispielsweise benötigt ein Wolkenkratzer, der sich in mehreren Kilometern Entfernung befindet trotz seiner Größe nur eine niedrig aufgelöste Diffuse Map. Ein nahes Objekt wie ein Hydrant, welcher sich im direkten Sichtbereich der Kamera befindet, braucht hingegen eine höher aufgelöste Textur. Für die Erstellung der Diffuse Maps von 3D-Objekten eignen sich Texturdatenbanken wie zum

57 Diffuse Map: Streufarben-Textur

58 Specular Map: Glanzpunkt-Textur

59 http://udn.epicgames.com/Three/TextureSupportAndSettings.html#Texture Resolution (Entnahmedatum: 02.01.2012)

Beispiel CG Textures[60] oder 3D Total[61]. Mittels bestehender Texturen lassen sich fotorealistische Objekte erzeugen. Sie lassen sich aber auch für andere Dinge einsetzen als sie ursprünglich gedacht waren. Eine raue Holztextur lässt sich beispielsweise einsetzen, um die Struktur eines Seils zu erzeugen. Für zusätzliche Varianz innerhalb der Textur sorgen auch verschiedene, miteinander verbundene Texturschichten. Dies kann in Bildbearbeitungsprogrammen zum Beispiel durch Ebenenmodi, welche die Überblendung zwischen mehreren Bildebenen definieren, erreicht werden. Hierdurch lassen sich zum Beispiel Metall sowie Rosttexturen kombinieren. Über die Maskierungsfunktion der jeweiligen Bildbearbeitungssoftware lassen sich zusätzlich noch die Einflussbereiche der Texturen ändern sowie Übergänge schaffen.[62]

4.3.3 Specular Map

Die Specular Map eines Objektes definiert die Farbe sowie Stärke des Glanzpunktes eines Objektes pro Pixel. Eine sehr helle Specular Map resultiert somit in einem sehr starken Glanzpunkt, dies zeigt Abbildung 12.

Abbildung 12: Auswirkungen der Specular Map-Helligkeit (unten) auf den Glanzpunkt des Objekts (oben) (Eigene Darstellung)

Es ist möglich, eine Specular Map aus einer bereits bestehenden Diffuse Map zu generieren. Hierbei muss beachtet werden, dass bestimmte Texturebenen der Diffuse Map

60 http://cgtextures.com/ (Entnahmedatum: 05.01.2012)

61 http://freetextures.3dtotal.com (Entnahmedatum: 05.01.2012)

62 Vgl. http://udn.epicgames.com/Three/CreatingTextures.html#Using Diffuse for Color, not Shading (Entnahmedatum: 02.01.2012)

nicht in der Specular Map berücksichtigt werden sollten. Beispielsweise handelt es sich dabei um Ambient Occlusion. Ein anderes Beispiel ist ein lackiertes gelbes Schild, auf dem sich schwarzer Text befindet. Würde die Specular Map aus einem solchen Schild heraus erzeugt werden, hätte nur die farbige Grundfläche des Schilds einen Glanzpunkt. Der schwarze Text des Schilds wäre hingegen matt. Der korrekte Aufbau der Specular Map wird in Abbildung 13 gezeigt. Die Auflösung der Specular Map ist abhängig von den Glanzeigenschaften des Objektes. Besitzt ein Objekt nur grobe Flächen, die sehr diffus scheinen, reicht es eine Textur mit der halben Auflösung der Diffuse Map zu verwenden. Bei Objekten mit sehr feinen glänzenden Stellen, wie beispielsweise Lackkratzer an einem Auto, ist eine hoch aufgelöste Specular Map nötig.[63]

Abbildung 13: Falsche Specular Map (links) und korrekte (rechts) (Eigene Darstellung)

4.3.4 Normal Maps

Normal Maps lassen sich mit gängiger Bildbearbeitungssoftware oder externen Programmen auch außerhalb von 3D-Animationssoftware erzeugen. Hierbei werden meist Plugins, wie das nVidia-Normal-Map-Plugin für Photoshop oder spezielle Programme, wie Crazybump[64] benötigt. Die manuelle Erzeugung einer Normal Map innerhalb dieser Programme setzt kein 3D-Modell, sondern eine Height Map voraus. Diese 8bit-Textur enthält die Höheninformationen, aus der die Normal Map generiert wird. Die Grundfarbe dieser Height Map ist ein mittlerer Grauton RGB (128,128,128). Hellere Farbbereiche der Height Map sorgen für eine Erhöhung, während dunklere Bereiche für eine Vertiefung sorgen. Eine Normal Map lässt sich anstatt aus einer Height Map auch aus einem Foto heraus generieren. Es treten jedoch hierbei Probleme auf, denn jedes Detail des Fotos wird anhand seiner Farbwerte analysiert und somit als Erhöhung oder Vertiefung dargestellt. Eine glatte Oberfläche mit einigen feinen Lackkratzern wird so-

63 Vgl. http://udn.epicgames.com/Three/TextureDefinedSpecularReflection.html#Overview (Entnahmedatum 02.01.2012)

64 Vgl. http://www.crazybump.com/ (Entnahmedatum: 02.01.2012)

mit falsch analysiert und die oberflächlichen Kratzer werden innerhalb der Normal Map zu starken Einkerbungen. Dasselbe Problem tritt bei Schrift auf. Dunkle Schrift auf einem hellen Hintergrund, wie beispielsweise bei einem Ortsschild, sorgt für eine starke Vertiefung innerhalb der Normal Map, obwohl das Objekt eigentlich eine glatte Oberfläche besitzt, da die verwendeten Programme den Text als Vertiefung interpretieren. Es ließ sich feststellen, dass die manuelle Erzeugung aus einer Height Map heraus für bessere Ergebnisse sorgt. Die Erzeugung aus einem Foto heraus bietet allerdings auch Vorteile. Sie ist sinnvoll, wenn feine Details wie zum Beispiel Stoffstrukturen erzeugt werden sollen, da hierbei aufgrund der Unregelmäßigkeit der Struktur die Ungenauigkeit weniger ersichtlich ist. Ebenfalls lassen sich diese nutzen, um zusätzliches Detail in eine bestehende Normal Map zu bringen.

Details, wie zum Beispiel Kratzer oder Poren, lassen sich gestaltn, indem eine vorhandem sie in eine vorhandene Normal Map hineinkopiert werden. Dies geschieht beispielsweise im Ebenenmodus "Ineinanderkopieren" im Bildbearbeitungsprogramm. Das manuelle Hinzufügen von Details ermöglicht eine höhere Kontrolle über die finale Normal Map.[65]

4.3.5 Weitere Texturen

Zur realistischen Darstellung von Objekten im UDK sind weitere Texturen nötig. Diffuse-, Specular- und Normal Map sorgen nur für eine grobe Definition des Aussehens eines 3D-Models. Zu den weiteren Texturen gehört beispielsweise eine Glossiness Map. Diese Map definiert für jedes Pixel die Härte des Glanzpunktes. Eine sehr helle Glossiness Map resultiert dementsprechend in einen sehr spitzen Glanzpunkt. Für Objekte mit gleichbleibender Glanzpunkthärte ist diese Textur nicht nötig, da der Glanzpunkt im UDK auch mittels einem Parameter gesteuert werden kann. Wenn der Glanzpunkt sich allerdings ändert, wie beispielsweise bei einem Stahlträger, der an einigen Stellen von Rost befallen ist, so ist eine solche Textur nötig, um eine realistische Darstellung zu gewährleisten. Mittels einer Height Map lassen sich im UDK spezielle Tiefeneffekte erzielen. Dies sorgt für einen starken 3D-Effekt bei Texturen. Das Verfahren wird Parallax Mapping genannt. Dabei werden die einzelnen Pixel der Textur in Relation zur Kameraposition verschoben. Die Stärke der Verschiebung definiert die Height Map.

65 Vgl. http://udn.epicgames.com/Three/CreatingNormalMaps.html#Overview (Entnahmedatum: 02.01.2012)

Abbildung 14: Parallax Mapping deaktiviert (links) und aktiviert (rechts) (Eigene Darstellung)

Eine Transparenzmaske wird im UDK dazu verwendet, um transparente oder halbtransparente Objekte zu erstellen. Abbildung 15 zeigt eine solche Maske. Die Verwendung von hart sowie weich maskierten Texturen ist möglich. Bei hart maskierten Texturen gibt es keine weichen Übergänge zwischen den verschiedenen Transparenzstufen. Weiche Masken arbeiten mit Farbverläufen. Die Verwendung der jeweiligen Maskierungstechnik muss im späteren Verlauf im Materialeditor des UDK gewählt werden.

Abbildung 15: InvertierteTransparenzmaske (Eigene Darstellung)

4.4 Skinning von Charakteren

4.4.1 Einleitung

Um Charaktere in 3D-Animationsfilmen zu animieren, werden diese mit einem virtuellen Skelett versehen. Dieses wird über das Skinning an das 3D-Modell gebunden und verformt es. Für ähnliche Charaktere bietet es sich hierbei an, dasselbe Skelett zu verwenden, da so im UDK auch Animationen anderer Charaktere funktionieren. Eine ein-

fache Möglichkeit ein zweibeiniges Skelett zu erstellen, ist die Verwendung eines vorgefertigten Bipeds (deutsch: Zweibeiner-Skelett), wie es in gewöhnlicher Animationssoftware verfügbar ist. Für nicht humanoide Charaktere oder Objekte müssen eigene Bone-Strukturen erstellt werden. Bones sind vergleichbar mit Knochen beim Menschen. Sie verformen bei Animation die Geometrie eines Objektes. Diese müssen für konventionelle Animationsfilme und auch für das UDK in einer hierarchischen Struktur zueinander aufgebaut sein. Zwei miteinander verbundene Bones haben immer eine Parent-to-Child-Beziehung (deutsch etwa: Eltern-Kind Beziehung) zueinander. Der Parent Bone beeinflusst hierbei die Bewegung des Child Bones, dieser kann nur um seinen eigenen Drehpunkt rotieren und kann wiederum der Parent Bone eines weiteren Child Bones sein.

4.4.2 Gewichtungsvergabe

Um ein Skinning auf ein 3D Modell anzuwenden, wird ein Modifikator beziehungsweise eine Funktion benötigt, welche es erlaubt, Bones an ein 3D Modell zu binden. Dieser Modifikator wird in der Regel Skin genannt. Dieser erstellt, sobald die Bones zugewiesen wurden, bereits Gewichtungen für jeden Bone. Diese Gewichtung lässt sich auf verschiedene Arten bearbeiten. Hierbei sind die Möglichkeiten programmabhängig. Es ist möglich, die Gewichtung über die Einflussbereiche der Bones zu ändern. Dies ermöglicht es, eine grobe Einstellung der Wirkungsbereiche der Bones vorzunehmen. Die Einflussbereiche lassen sich in einigen Programmen auch über ein Zeichenwerkzeug auf die 3D-Geometrie malen. Dies wird als Weight Painting bezeichnet und erlaubt eine hohe Genauigkeit des Skinnings. Die Stärke der Beeinflussung wird hierbei optisch durch farbige Flächen angezeigt. Noch exakter lassen sich die Gewichtungen über die manuelle Vergabe pro Knotenpunkt festlegen. Hierbei wird die Stärke zwischen 0 und 100 Prozent oder durch einen Wert zwischen 0 und 1 angegeben. Bei der Verwendung im UDK ist zu beachten, dass ein Knotenpunkt nur von maximal vier Bones beeinflusst werden kann. Alle weiteren Bones, die diesen Punkt bei der Erstellung im Animationsprogramm beeinflussen, werden vom UDK ignoriert. Dies schränkt vor allem bei Charakteren ein, da bei diesen ein Körperteil oft von vielen Bones beeinflusst wird.

4.4.3 Bonesets

Im UDK lassen sich beliebige Skelette (Bonesets) verwenden. Es ist hierbei egal, ob es sich um Bones aus Nullobjekten, 3D-Modellen oder direkten Bone-Objekten handelt.

Falls beliebige Objekte über einen Skinning-Modifikator mit einem Objekt verbunden sind, wirken diese als Bones. Für Charakteranimationen ist es wichtig, dass Charaktere, die der gleichen Animationsgruppe zugehörig sind und somit auch die gleichen Animationen nutzen, über dasselbe Boneset verfügen. Ist dies nicht der Fall, muss für jeden Charakter ein eigenes Animationsset erstellt werden.

Um Charaktere mit verschiedenen Skeletten mit den gleichen Animationen ansprechen zu können, erweisen sich Verknüpfungen, wie zum Beispiel Parent-to-Child als nützlich. Mit diesen können einzelne Bones an ein Steuerskelett gebunden werden. Dieses Steuerskelett kann für eine Gruppe oder alle Charaktere gleich sein. Im späteren Verlauf muss nur das Steuerskelett animiert werden. Die in der Hierarchie darunterliegenden Bones beziehen ihre Translation und Rotation von dem mit Verknüpfungen angebundenen Steuerskelett. Diese Arbeitsweise ist nützlich, wenn ein oder mehrere Charaktere bereits mit vollständigem Skinning verfügbar sind und eine Anpassung zu zeitaufwendig ist. Ein Problem der Arbeit mit Verknüpfungen ist, dass sich die Anzahl der verwendeten Bones enorm erhöht. Bei der Arbeit innerhalb von 3D-Bearbeitungsprogrammen empfiehlt es sich deshalb, die einzelnen Bones in Ebenen beziehungsweise Ordner zu sortieren, um die Arbeitsumgebung übersichtlich zu halten. Dies ermöglicht ein effektiveres Arbeiten. Über Parent-to-Child an das Skelett angebundene 3D-Models werden im UDK als Bones behandelt. Dies erweist sich beispielsweise bei Objekten als nützlich, die nachträglich im UDK animiert werden sollen. Beispielsweise ist dies bei Augen möglich. Diese lassen sich über sogenannte Skeletal Controls im UDK ansprechen.[66]

4.4.4 Bones für die Kleidungssimulation

Die Unreal-Engine verfügt über eine eigene integrierte Stoffsimulation. Diese ermöglicht es, realistisch fallende Tücher oder im Wind wehende Flaggen zu erstellen. Auch Kleidungsstücke von Charakteren, wie zum Beispiel Umhänge oder Röcke, lassen sich damit automatisch animieren. Die Berechnung erfolgt je nach Hardwareausstattung des Computers entweder über eine nVidia PhysX taugliche Grafikkarte oder den Prozessor. Da die Berechnung sehr rechenintensiv ist, wird der Prozessor hierbei entsprechend stark belastet. Problematisch für die automatische Stoffsimulation sind Szenen, in denen eine hohe Interaktion des Charakters mit dem Stoff vorherrscht. Da die Simulation bei jedem Durchlauf anders abläuft, kann das Erreichen korrekter Simulationsergebnisse einige Zeit in Anspruch nehmen. Hierfür bietet es sich an, alles manuell mit-

66 http://udn.epicgames.com/Three/UsingSkeletalControllers.html#Overview (Entnahmedatum: 02.01.2012)

tels Bones zu animieren. Der einmalige Mehraufwand ist hierbei zwar höher, jedoch ist das Endresultat immer exakt und ressourcensparender. Hierbei erstellt der Animator optisch ausreichende Kleidungsanimationen während er die volle Kontrolle über den Animationsprozess besitzt.[67] Bei einer manuellen Stoffsimulation muss wie beim Skinning von Charakteren vorgegangen werden. Es werden hierbei weiche Übergänge zwischen den einzelnen Einflussbereichen benötigt, da sonst harte Bruchkanten entstehen. Eine hohe Anzahl von Bones erschwert das Animieren zudem.

4.5 Export von Szenenobjekten in das UDK

4.5.1 Allgemeines

Um das Skinning von Charakteren und Objekten in das UDK zu exportieren, muss der Modifikator in der Hierarchie über allen anderen stehen. Dies hat zur Folge, dass es nicht möglich ist, einen Subdivision-Modifikator nach dem Skinning anzuwenden. Dadurch ergeben sich Einschränkungen im Detailgrad der Modelle. Falls dennoch ein höherer Detailgrad benötigt wird, muss das gesamte Modell inklusive Unterteilungen mit Skin-Modifikator versehen werden. Dies erzeugt einen deutlich höheren Arbeitsaufwand, da sich die Anzahl der Knotenpunkte massiv erhöht.

4.5.2 Exportformate

Das UDK akzeptiert eine Reihe von Formaten für den Import von Charakteren und Objekten. Ein altes Dateiformat ist der Ascii Scene Exporter, kurz ASE. Dieses Format ermöglicht es nur, statische Objekte zu exportieren. Texturen lassen sich nicht einbinden.

Die offiziellen von Epic Games veröffentlichten Richtlinien zur Erstellung von Objekten empfehlen das Autodesk FBX-Exportformat[68]. Dieses Dateiformat ermöglicht es, statische Objekte sowie Charaktere zu exportieren. Mittels Autodesk 3Ds Max ist es möglich, ganze Materialien sowie Texturen zu exportieren. Hierbei ist folgendes zu beachten:

67 Vgl. Faking Dynamics of Cloth Animation for Digital Films, Seite 240

68 FBX: Exportformat für 3D-Objekte, vgl. http://udn.epicgames.com/Three/FBXPipeline.html (Entnahmedatum: 02.01.2012)

- Es werden beim Import nur Materialien des Typs "Standard" oder „Multi/Sub[69]“ importiert.[70] Unterstützte Maps sind die Diffuse-, Specular-, Emissive-, Glossiness- und Bump Map mit Normal-Map-Slot.

- Durch den Export im FBX-Format wird im Gegensatz zum früheren Arbeitsablauf mit ASE-, PSK- oder PSA-Format der Import vereinfacht[71]. Materialien und Texturen werden beim Import automatisch mit in das UDK geladen. Hierdurch entfällt die manuelle Erstellung der Materialien. Anpassungen dieser sind weiterhin möglich.

- Seit September 2011 wird nur noch die Autodesk FBX 2012 Version für den Import in das UDK unterstützt.[72]

69 Multi/Sub-Material: Material, welches in mehrere Untermaterialien unterteilt ist

70 Vgl. http://udn.epicgames.com/Three/FBXMaterialPipeline.html#Material Support (Entnahmedatum: 02.01.2012)

71 ASE, PSK und PSA: veraltete Exportformate für 3D-Objekte

72 Vgl. tp://udn.epicgames.com/Three/FBXPipeline.html (Entnahmedatum: 02.01.2012)

5 Animationen für das UDK

Die Weiterverarbeitung von Charakter-Animationen im UDK ist auf die Anwendung in Spielen angelegt. Dies wird deutlich in den Einschränkungen, die es für Animationen im UDK gibt. Die Animation von physikalischen Elementen, wie zum Beispiel einem Partikelsystem, lassen sich direkt im UDK vornehmen und sind funktional. Es ist mit Einschränkungen möglich, Stoffe zu simulieren. Hierbei wird eine Physik-Simulation angewandt, die nicht auf Kleidung ausgelegt ist. Die Darstellung von Emotionen kann mit Morph Targets erfolgen. Morph Targets überblenden zwischen mehreren vordefinierten 3D-Modellen gleichen Aufbaus. Dies schränkt die Varianz der Emotionsvisualisierung ein, ermöglicht jedoch einen schnellen und effizienten Arbeitsablauf.

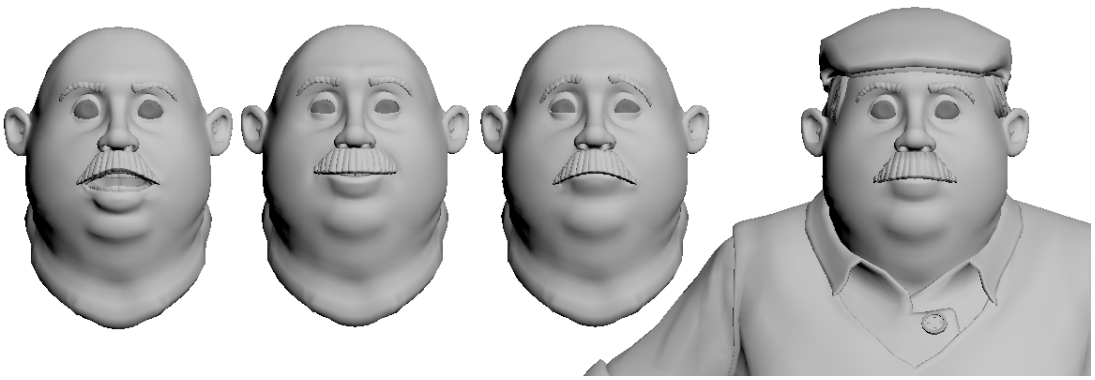

Abbildung 16: verschieden Modelle aus denen Morph Targets generiert werden (links) und Referenzmodell (rechts) (Eigene Darstellung, Charakter von Pixable)

5.1 Gestik und ganzkörperliche Bewegungen

Grundsätzlich ist es nicht möglich, Charakter-Animationen im UDK zu erstellen oder zu variieren. Die Entwicklungsplattform erlaubt lediglich, Bewegungen des ganzen Charakters zu erstellen, Morph Targets anzusteuern, Kopfdrehungen zu realisieren und Animationen zu überblenden. Deshalb sollten die Animationen in einem geeigneten Programm, wie MotionBuilder[73] oder 3ds Max erstellt werden. Die Animationen müssen so konzeptioniert werden, dass die Bewegung der Bones für jeden Charakter einzeln erstellt und exportiert werden können. Denn beim Import von Animationen werden nur

73 http://www.autodesk.de/adsk/servlet/pc/index?id=15013088&siteID=403786 (Entnahmedatum: 05.01.2012)

die Animationen und nicht die Szenerie oder der Charakter importiert.[74] In Folge dessen lassen sich Animationen von großen Menschengruppen nur umständlich exportieren. da für jeden Menschen eine eigene Datei erstellt werden muss, welche einzeln exportiert werden muss.

5.1.1 Einschränkungen bei der Erstellung von Animationen

Beim Erstellen von Animationen gibt es einige Einschränkungen. Pro Charakter können mehrere Animationsspuren in das UDK importiert werden, hierfür ist jedoch eine bestimmte Benennung nötig.[75] Dies ist in der Organisation der Erstellung der Animationen zu beachten. Es ist nur möglich, geplottete oder gebackene Animationen zu importieren.[76] Das heißt, dass die Animationen auf die Bones der Figur gebracht werden müssen. In Folge dessen sind die Animationen im Nachhinein nicht mehr variabel, da für jedes Einzelbild und jeden Bone ein Keyframe angelegt wird. Eine Bearbeitung der Bewegung ist damit nur sehr mühsam und umständlich möglich. Die Animationen sollten immer vom Null-Punkt im Koordinatensystem ausgehen, um die weitere Bearbeitung im UDK zu vereinfachen. Ist dies nicht der Fall, wird die Platzierung des Charakters erschwert, da der Pivot[77] im Koordinatenursprung verzeichnet ist. Führt der Charakter also beispielsweise erst eine Animation aus, die im Koordinatenursprung erstellt wurde und danach eine andere, die nicht im Null-Punkt beginnt, wird der Charakter zwischen den beiden Animationen sprunghaft die Position verändern. Folglich ist die Figur nicht mehr spielbar, da der Spieler den Null-Punkt der Figur bewegt. Auch bei der Positionierung und Positionsveränderung im UDK ergibt sich dieses Problem.

Für bestimmte Bewegungsabläufe bietet es sich an, loopbare Animationen zu erstellen. Dies bedeutet, dass die Animation beliebig oft hintereinander abgespielt werden kann. Der letzte Frame der Animation muss also auf den ersten Frame der Animation abgestimmt sein. Abbildung 17 verdeutlicht den Aufbau einer solchen Animation. Hierbei sind vor allem Einzelbild 1 und Einzelbild 8+ relevant. Diese sollten entsprechend auf der Stelle ausgeführt werden. Außerdem ist es für eine loopbare Animation wichtig, dass die Posen in gleichen Abständen ablaufen sollten. Hat die Animation beispiels-

74 Vgl. http://udn.epicgames.com/Three/FBXAnimationPipeline.html#Exporting Animations from 3D Apps (Entnahmedatum: 10.01.2012)

75 Vgl. http://udn.epicgames.com/Three/FBXAnimationPipeline.html#Exporting Animations (Entnahmedatum: 02.01.2012)

76 Vgl. http://udn.epicgames.com/Three/FBXSkeletalMeshPipeline.html#Export Mesh (Entnahmedatum: 02.01.2012)

77 Pivot: Objektursprung bzw. Objektnullpunkt

weise zwei Posen, also eine Startpose, die gleichzeitig die Endpose ist, sollte die mittlere Pose zeitlich genau in der Mitte liegen.

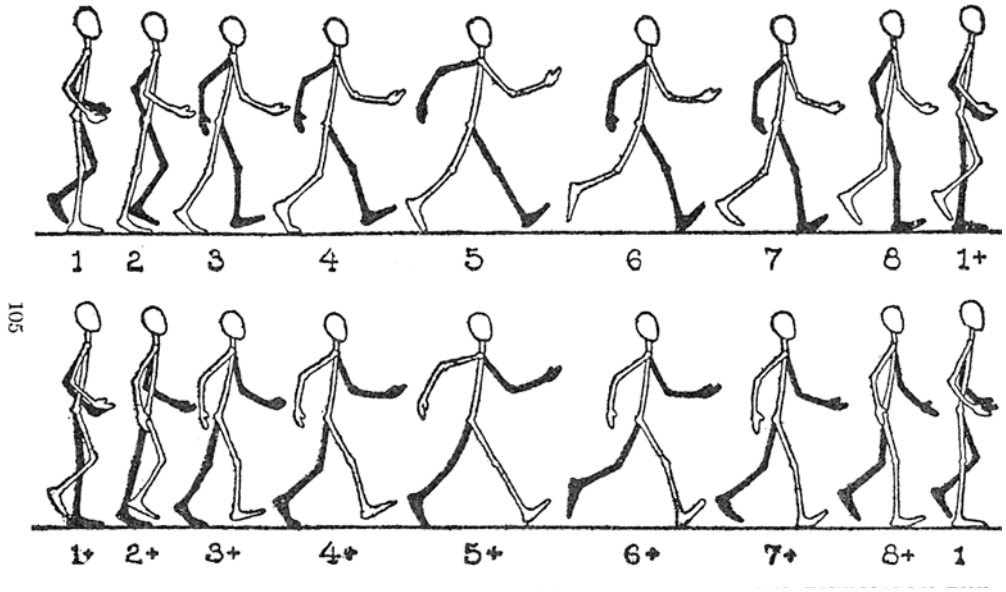

Abbildung 17: Frameweise Darstellung eines Walkcycle[78]

Ein Beispiel hierfür wäre die Animation des Laufens. Ein Durchgang der Animation wird erstellt. Start und Endpose müssen aufeinander abspielbar sein. Wenn die mittlere Pose nicht in der Mitte der Zeit liegt, wird der Charakter beim mehrfachen Abspielen der Animation humpeln. Man kann feststellen, dass das Loopen von Animationen in Verbindung mit der Blickrichtungsanpassung großen Arbeitsaufwand erspart, da Animationen beispielsweise mit der Blickrichtungsanpassung in Matinee variiert werden können und nicht jede Variation einzeln importiert werden muss. Die Blickrichtungsanpassung erlaubt die Rotation eines Bones mit einem Kontrollpunkt. Somit lässt sich beispielsweise der Kopf oder die Augen drehen. Interaktionen zwischen Charakteren sind durch den getrennten Export mit einigen Hindernissen verknüpft. Beim Erstellen des Testfilms konnte man feststellen, dass Animationen immer nach vorn, also mit Hauptblickrichtung zur positiven z-Achse exportiert werden sollten. So werden sie im UDK auch mit der korrekten Blickrichtung importiert. Dabei ist zu beachten, dass die Charaktere im UDK genau platziert werden müssen. Bei der Erstellung des Kurzfilms wurde festgestellt, dass wenn sich Charaktere berühren sollen oder mit der Umgebung interagieren, dies durch die Platzierung erschwert wird. Dies wird in Abbildung 18 deutlich. Die Hand des Charakters ist im 3D-Bearbeitungsprogramm korrekt positioniert. Im UDK bewegt sich die Hand in der Animation durch den anderen Charakter.

78 http://www.juliecallahan.me/walkcycle.jpg (Entnahmedatum: 02.01.2012)

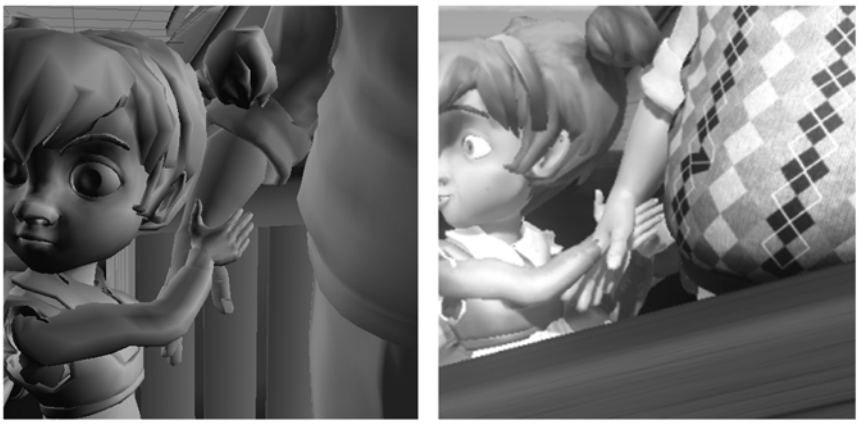

Abbildung 18: Korrekte Animation bei der Erstellung (links) Animation im UDK mit fehlerhafter Handposition (rechts) (Eigene Darstellung, Charaktere von Pixable)

Bei Animationen, bei denen zwei Charaktere abhängig voneinander interagieren, ist der Export komplexer, da alle Charakter-Animationen einzeln und ohne Charakterisierung exportiert werden müssen. Dies ließ sich beim Erstellen der Animationen für den Film feststellen. Hierbei ist ein Charakter immer Akteur und der Gegenpart wird durch Parent-to-Child an die Bewegungen des Akteurs gebunden. Der Charakter oder das Objekt, bei dem die Bewegung erzwungen wird, muss immer vor der Bewegungsquelle exportiert werden. Hierbei ließ sich feststellen, dass es von Vorteil ist, den Charakter am Ort der Pose zu plotten und die Animation dann auf einen Referenz-Charakter im Animationsprogramm zu laden. So gehen an andere Akteure gebundene Bewegungen nicht verloren. Danach kann dieser exportfähig gemacht werden. Der Export lässt sich optimal mit dem Format FBX 2011 realisieren. Hierbei sollte, wenn möglich, nicht komprimiert werden, um die bestmögliche Animations-Qualität zu erhalten. Während der Arbeit am Kurzfilm ließ sich feststellen, dass FBX-Formate vor dem Exporter 2011 nur mangelhaft unterstützt werden und der Import solcher Formate eine fehlerhafte Darstellung der Animation aufweist. So werden zum Beispiel Bewegungen teilweise ohne erkennbares Muster ignoriert und nicht importiert. Beim Export ergibt sich das Problem, dass humanoide Charaktere in der Unreal Engine ihren Mittelpunkt der Bones in der Hüfte haben.[79] In handelsüblichen Animationsprogrammen liegt der Mittelpunkt meist zwischen den Füßen. Deshalb werden Animationen teilweise fehlerhaft importiert. So kann man feststellen, dass ein Einknicken der Beine am Kniegelenk im UDK nicht dargestellt wird, obwohl es in der Datei vorhanden ist. Aus den genannten Problemen folgt, dass der Animationsimport ins UDK nur teilweise realisierbar ist, da es unvorher-

79 Vgl. http://udn.epicgames.com/Three/FBXSkeletalMeshPipeline.html (Entnahmedatum: 02.01.2012)

sehbar zu unbrauchbaren Ergebnissen kommen kann und der Arbeitsablauf durch eine solche Fehleranfälligkeit eingeschränkt wird.

5.1.2 Weiterverarbeitung von Animationen im UDK

Im UDK müssen Interaktionen mit Gegenständen über Sockets realisiert werden. Dies bedeutet, dass ein Punkt am Charakter festgelegt wird, an dem das Objekt befestigt wird. Der Charakter kann so zum Beispiel eine Fackel halten. Die Darstellung des Aufnehmens von Objekten wird so erschwert, da ein Objekt immer an den Sockel heranspringt. Daher sollten solche Animationen in der Konzeption vermieden oder nicht dargestellt werden, beispielsweise indem im Moment des Greifens geschnitten wird. Alternativ können auch beide Objekte, also beispielsweise die aufnehmende Hand und das Zielobjekt manuell animiert werden, was aber in einem hohen Mehraufwand resultiert. Grundsätzlich ist es möglich, Animationen als PSA-, FBX- oder als Collada-Animationsdateien direkt in das AnimSet zu importieren.[80] Das AnimSet ist eine Bibliothek, in der alle Animationen für einen Charakter gespeichert werden. Für den Charakter wird im UDK ein AnimSet erstellt, in das alle Animationen exportiert werden.[81] Für den funktionierenden Import von Animationen ist es nötig, dass in der Animationsdatei die gleichen Bones verzeichnet sind wie auch im Charakter. Hierbei ist es absolut nötig, dass die Bones jeweils gleich benannt werden. Wenn in der Quelldatei Bones fehlen, ist es nicht möglich. die Animation zu importieren. Einzelne Bones, die in der Animationsdatei vorhanden sind, aber im UDK-Charakter fehlen, werden ignoriert und die Animation kann in der Regel importiert werden. Wenn die Charaktere das gleiche Skelett besitzen, können AnimSets wahlweise auch mehrfach auf diese angewendet werden.[82] Dies kann bei Charakteren mit gleichem Wesen, also beispielsweise einer Mehrzahl von Kindern, zu einer hohen Arbeitsersparnis führen und die optische Qualität bleibt fast unbeeinflusst. Es bietet sich aber auch an, die vorhandenen Animationen zu variieren, damit sich nicht alle Charaktere synchron bewegen. So besteht im UDK die Möglichkeit, Animationen zu überblenden und somit beispielsweise beim Laufen die Armbewegung zu variieren.[83] Das AnimSet wird mit dem Charakter und dem AnimTree verknüpft und die Animationen werden im AnimTree mit BlendNodes an den Charakter gebunden und basierend auf den Nodes definiert. Der AnimTree dient dazu, im Spiel zu defi-

80 Vgl. http://udn.epicgames.com/Three/CreatingAnimations.html (Entnahmedatum: 02.01.2012)

81 Vgl. http://udn.epicgames.com/Three/ImportingAnimationsTutorial.html#Importing Animations (Entnahmedatum: 02.01.2012)

82 Vgl. http://udn.epicgames.com/Three/CreatingAnimations.html#Development Tools (Entnahmedatum: 02.01.2012)

83 Vgl. http://udn.epicgames.com/Three/AnimationOverview.html#Animation Blending (Entnahmedatum: 05.01.2012)

nieren, was der Charakter in welcher Situation tun soll. Die Fallunterscheidung der Situationen werden als BlendNodes bezeichnet. Es ist im UDK beschränkt möglich, die Rotation von Bones anhand eines Zielobjektes in Matinee zu animieren. Dabei wird aber nach unserem Kenntnisstand immer der Spieler als Zielobjekt definiert. Dieser ist allerdings nicht in Matinee animierbar. Als Problem der Lösung ist es möglich, eine Box mit intrudierter Deckfläche zu erstellen. Diese Box wird im Level erstellt und bekommt als Attribut Undurchlässigkeit zugewiesen und der Spieler-Startpunkt wird darin platziert. Die Box kann mit einem MovementTrack[84] versehen und so animiert werden. Der in der Box befindliche Spielerstart wird so indirekt bewegt. Durch diesen Umweg lässt sich dann die Rotation des Bones steuern. Durch die Verwendung der Box ist es schwierig, die Kopf- und Augenbewegung genau zu steuern, da diese Bewegungen einerseits nicht im Vorschaufenster angezeigt werden und andererseits die Box und der Spieler durch ihre Dimensionierung kein genaues Ziel darstellen können.

5.1.3 AnimTree

Der AnimTree eines Charakters definiert, wie im Spiel zwischen zwei Animationen überblendet werden soll. Dies ist ein nodebasiertes System. Hierbei wird mit sogenannten BlendByNodes auf verschiedene Zustände eingegangen und die passende Animation für den Charakter definiert. So ist es zum Beispiel möglich, je nach Geschwindigkeit, Idle[85], umgebendem Medium oder Tastendruck zu blenden. Dies ist nötig, um den Charakter im Spiel steuerbar zu machen.[86]

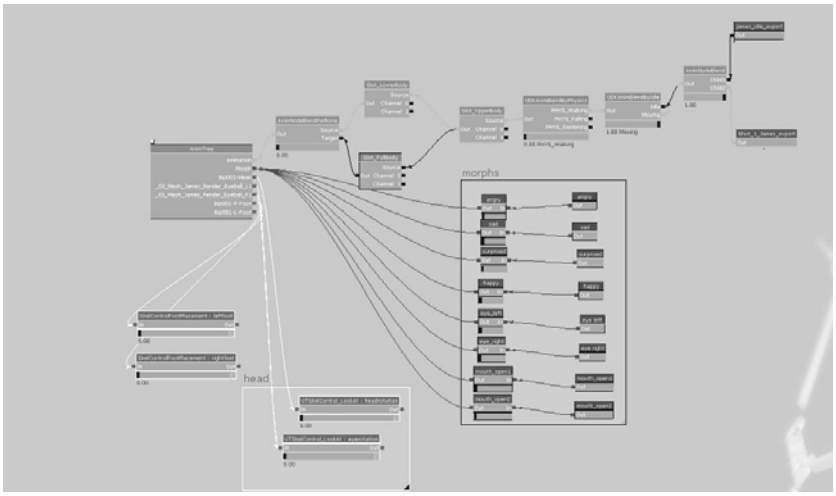

Abbildung 19: AnimTree eines Charakters (schematisch, eigene Darstellung)

84 Movementtrack: Definiert die Position des Charakters im Level und ist animierbar

85 Idle: „Nichts-tuen“ des Spielers, Animation für den Fall, dass der Spieler keine Eingabe macht

86 Vgl. http://udn.epicgames.com/Three/AnimTreeEditorUserGuide.html (Entnahmedatum: 02.01.2012)

Der AnimTree eines Spieler-Charakters resultiert meist in einer komplexen Baumstruktur, da viele Animationen unterschiedlich überblendet werden müssen. Für die Arbeit in Matinee ist ein AnimTree zwar nötig, aber er muss nur aus Grundelementen bestehen. Die Animationen müssen sich nicht zwangsläufig darin befinden. Weiterhin werden MorphWeights für die einzelnen Morph Targets im AnimTree vergeben. Dies erfolgt unabhängig von den Blend-Nodes. Bei den MophWeights ist es möglich, Werte über 100% darzustellen. Dies sollte aber immer bewusst getan werden, da die Morph Targets bei Werten über 1 übermäßig verstärkt werden. Im AnimTree werden des Weiteren die Kopfdrehung und das Foot-Placement[87] definiert.

5.2 Emotionen und Lippenbewegungen

Im UDK gibt es grundsätzlich zwei Möglichkeiten, Emotionen und Lippenbewegungen bei menschlichen Charakteren darzustellen:

- mit einem Face-Rig, also einem Skelett, mit dem das Gesicht gesteuert wird, und dem Plug-In FaceFX, welches Sprache automatisch lippensynchronisiert
- mit Morph Targets, also Posen, zwischen denen überblendet wird.

Das Face-Rig erlaubt es, abwechslungsreiche und ansprechende Animationen zu realisieren.Mittels Morph Targets kann eine schnelle Arbeitsweise garantiert werden, jedoch ist die Varianz der Gesichtsanimationen beschränkt auf die vom 3D-Künstler definierten Morph Targets.

5.2.1 Emotionsdarstellung mittels Morph Targets

Morph Targets sind Deformationen des Models, bei denen Knotenpunkte verschoben werden. Verschiedene Varianten mit Deformationen werden mittels Modifikator auf das Referenz-Modell gebracht welches dann verformt wird. Hierbei wird die vektorielle Bewegung der Eckpunkte auf das Objekt gespeichert.[88] So ist es möglich, das Morph Target zwischen 0 und 100% anzeigen zu lassen. Die Stärke der Überblendung bezeichnet man als Morph Weight. Moph Targets lassen sich im UDK in ein Morph Targets Set einfügen und dann über einen AnimTree und mit Matinee ansteuern.[89] Morph Targets

87 Foot-Placement: Funktion, die es ermöglicht, dass die Füße nicht durch den Boden stoßen oder darüber schweben.

88 Vgl. Autodesk 3ds Max 2011 Help, Seite 1464

89 Vgl. http://udn.epicgames.com/Three/MorphTargets.html (Entnahmedatum: 02.01.2012)

lassen sich in einem 3D-Bearbeitungsprogramm wie 3ds Max oder Maya erstellen. Um perfekte Qualität und Varianz wie im konventionellen 3D-Animationsfilm zu erhalten, wäre eine Vielzahl von Emotionen nötig. Wenn diese mit Morph Targets abgedeckt würden, müssten alle Regungen im Gesicht dargestellt werden und die Emotionszustände müssten von Situation zu Situation variieren. Wenn effiziente und schnelle Arbeitsabläufe hohe Priorität haben, reicht es aus, einfache emotionale Zustände zu erschaffen. Beim Erstellen von Testszenen für den Kurzfilm wurde festgestellt, dass sich folgende Auswahl für eine akzeptable Mimik anbietet:

Morph Target	**Augenlider**	**Augenbrauen**	**Wangen**	**Mund**
Fröhlich (happy)	Leicht zusammengekniffen	In der Mitte nach oben	Leicht nach oben	Mundwinkel weit nach oben, Mund leicht geöffnet
Traurig (sad)	Leicht geschlossen	Nach oben (in der Mitte am stärksten	-	Mundwinkel weit nach unten
Erschrocken (surprised)	Weit geöffnet	Nach oben	-	Geöffnet
Wütend (angry)	Zusammengekniffen	In der Mitte stark verengt und nach unten zeigend	-	Geschlossen

Tabelle 1: Morph Targets für Emotionen.

Abbildung 20: Morph Targets für die Darstellung von Emotionen (Eigene Darstellung, Charakter von Pixable)

Es ließ sich feststellen, dass Emotionen unrealistisch stark erstellt werden sollten. So ergibt sich eine größere Varianz. Die Mangelnde Auswahl an Morph Targets wird also durch Varianz in der Stärke ausgeglichen. Außerdem ist es möglich, mehrere Morph Targets miteinander zu kombinieren. Im Praxistest wurde deutlich, dass folgende Morph Targets nötig sind um Varianz in der Mimik des Charakters zu schaffen:

- Augenschließung: das linke und das rechte Auge werden jeweils als einzelne Morph Targets erstellt
- Mundöffnung: es werden mindestens zwei Morph Targets mit unterschiedlichen Mundöffnungen erstellt.

5.2.2 Lippensynchronisation mittels FaceFX-Studio

FaceFX-Studio ist ein Programm, in dem Sprache auf Phoneme[90] analysiert wird und daraufhin die Lippenbewegung eines Charakters synchronisiert wird. Hierfür ist es nötig, die einzelnen Phoneme in einer Zuordnungstabelle zuzuweisen. Zudem eignet sich FaceFX, um mittels Face-Rig Emotionen zu animieren. Hierbei ist es möglich, mit einem Face-Rig Gesichtsposen für die Phoneme festzulegen. Allerdings können auch Morph Weights verwendet werden. Mit dem FaceFX-Studio-Plug-In im UDK lässt sich ein FaceFX-Asset erstellen für einen Charakter erstellen. In diesem Asset lassen sich FaceFX-AnimSets erstellen. In dem FaceFX-Animset werden alle Animationen des jeweiligen Charakters verzeichnet.[91] Mittels Morph Targets lässt sich eine Zuordnungstabelle erstellen, bei dem die orthographischen Realisierungen der Phoneme der jeweiligen Lippenpose zugewiesen werden. Sprachdateien (sogenannte SoundCues) lassen sich in FaceFX importieren und automatisch analysieren. Hierbei kann zur Unterstützung der Text mit analysiert werden. Zusätzlich lassen sich die Sprechgeschwindigkeiten schnell, langsam, normal und automatisch als Vorgabe setzen. Daraufhin wird automatisch eine Abfolge von Phonemen erstellt und der Charakter spricht. Diese Abfolge kann im Nachhinein an den SoundCue angepasst werden.[92] Durch die Anwendung von FaceFX in den Testszenen wurde deutlich, dass dieses Mittel für effiziente und funktionale Lippensynchronisation dient.

90 Phonem: kleinste Einheit der Sprache

91 Vgl. http://udn.epicgames.com/Three/FaceFX.html (Entnahmedatum: 02.01.2012)

92 Vgl. http://udn.epicgames.com/Three/IntroductionToFaceFX.html#Introduction to _FaceFX (Entnahmedatum: 02.01.2012)

5.3 Animation von Kleidung und Stoffen

Das UDK verfügt über eine Stoff-Simulation. Mit dieser werden in Echtzeit Bewegungen von Stoffen und Kleidung dargestellt. Es können hierbei Variablen wie Dicke und Steifigkeit eingegeben werden.[93] Der Praxistest zeigte, dass die Stoffsimulation nur teilweise mit Kleidung funktioniert. Es hat sich herausgestellt, dass bei schnellen Bewegungen des Charakters in jedem Einzelbild eine abwechselnde Kollision mit der Figur und der Umgebung erfolgt. Dadurch ergibt sich ein optisch nicht zufriedenstellendes Ergebnis der Kleidungssimulation. Sitzt der Charakter beispielsweise, flackert der Stoff, da er von dem Untergrund und vom Charakter abwechselnd in jedem Einzelbild abgestoßen wird. Für den Kurzfilm wurde deshalb auf die Stoffsimulation bei Charakteren verzichtet. Die Kleidung wurde mit Bones versehen und von Hand animiert. Dies ist allerdings gerade bei dynamischen schnellen Bewegungen sehr aufwändig. Soll die Stoffsimulation für andere weniger dynamische Animationen trotzdem angewendet werden, muss der Charakter doppelt importiert werden: einmal mit Bones für Kleidung, einmal ohne. Bei Animationen, die weniger dynamisch oder schnell sind und keine Interaktionen mit der Umgebung aufweisen, zeigt der Praxistest, dass die Simulation von Stoffen problemlos funktioniert. Ein Beispiel für eine solche Animation ist das Laufen. Die Kleidung kollidiert nur mit Körperteilen und die Simulationsergebnisse sind nutzbar. Optimalerweise sollte die nur teilweise funktionale Kleidungssimulation bereits in der Konzeption des Charakters und bei der Erstellung von Animationen beachtet werden. Die Stoff-Simulation eignet sich für einfache Stoffe, wie ein Segel oder einen Vorhang, der sich vom Wind beeinflussen lässt. Die Verwendung von Stoff-Simulation an einem Charakter ist also nur begrenzt möglich, wohingegen die Verwendung an unbeweglichen Objekten weitgehend problemlos funktioniert.

5.4 Partikelsysteme

Im UDK ist es möglich, Partikelsysteme zu erstellen. Entsprechende Materialien können mit Parametern wie Geschwindigkeit oder Rotation als Partikel wiedergegeben werden. Hierbei können verschiedenste Parameter das Partikelsystem beeinflussen. Die Partikelsysteme werden immer als Fläche dargestellt, da sie aus Materialien erstellt werden . Diese Fläche wird aber automatisiert immer in Blickrichtung des Spielers oder der Kamera gedreht, so dass nicht auffällt, dass es sich bei dem System nur um

93 Vgl. http://udn.epicgames.com/Three/PhysXClothReference.html (Entnahmedatum: 02.01.2012)

Flächen handelt.[94] Rauch wird beispielsweise mit Sets aus verschiedenartigen Rauchwolken erstellt. Diese Sets werden mit dem Materialeditor vorbereitet. Danach werden sie mit entsprechenden Parametern versehen. Hierbei ist die Möglichkeit gegeben, für ein Partikelsystem mehrere Partikel aus unterschiedlichen Materialien mit unterschiedlichen Parametern zu erstellen und diese dann zu vermischen. So lassen sich unterschiedliche Eigenschaften für Quellen in einem Partikelsystem erstellen.[95] Es ist möglich, Partikelsysteme mit einem Sockel an einen Charakter oder ein mit Bones versehenes Objekt zu binden. So lässt sich beispielsweise ein Feuer an eine Fackel binden. Die Partikelsysteme können sich von ihrer Umwelt beeinflussen lassen und reagieren physikalisch korrekt. Wird also die Fackel aus dem Beispiel bewegt, wird sich das Partikelsystem mit einer physikalischen Trägheit nachziehen. Der Ursprung des Partikelsystems bleibt dabei immer fest am Sockel.

Abbildung 21: Fackel mit angebundenem Partikelsystem (Eigene Darstellung, Charakter von Pixable)

94 Vgl. http://udn.epicgames.com/Three/ParticleSystemReference.html#Overview (Entnahmedatum: 02.01.2012)

95 Vgl. http://udn.epicgames.com/Three/CascadeUserGuide.html (Entnahmedatum: 02.01.2012)

6 Erstellung des Settings

6.1 Allgemeines

Das UDK bietet vielfältige Möglichkeiten, verschiedenste Landschaften und Umgebungen zu erzeugen, die dabei ressourcenschonend berechnet werden können. Hierbei können sowohl urbane Szenerien, Innenräume, als auch weiträumige Landschaften erstellt werden. Mit den Werkzeugen des UDK bieten sich Möglichkeiten, sowohl Terrains mit ausschweifender Vegetation zu erstellen, als auch urbane Szenen mit viel Architektur. Die Szenen lassen sich ressourcenschonend darstellen, da die UDK-Werkzeuge auf das Echtzeit-Rendering abgestimmt sind.

6.2 Landschaftserstellung

Im UDK gibt es die Möglichkeit, mit einem Werkzeug Landschaften zu gestalten. Dieses Tool bietet vielfältige Möglichkeiten und es empfielt sich, das Terrain-Tool einzusetzen, denn nur so ist eine funktionierende Lichtberechnung für die Landschaft möglich. Landschaften können maximal 512k x 512k Unreal Einheiten groß sein. Empfohlen werden Ausmaße von maximal 1024 x 1024 Unreal Einheiten.[96] Die Tessalation, also Unterteilung des Terrains kann unterschiedlich stark vorgenommen werden. Jedoch bringt eine große Landschaft und eine hohe Unterteilung auch eine langsamere Lichtberechnung und höhere Rechenzeiten bei der Bearbeitung mit sich.

Neben der Erstellung im UDK ist es möglich, Height-Maps zur Erstellung von Terrains zu importieren. Diese können in Bildbearbeitungsprogrammen wie Photoshop erstellt werden. Auch mit Landschaftsdesign-Programmen wie Terragen[97] können Height-Maps erstellt werden. Es ist auch möglich, in 3D-Bearbeitungsprogrammen wie 3ds Max Landschaften zu modellieren und basierend auf diesem Modell eine Heightmap zu erstellen. Pro Pixel wird eine Unreal-Einheit erstellt.[98] Die Height Map sollte in den Ausmaßen immer ein Pixel größer sein, als die gewünschte Terrain-Größe. Soll das Ter-

96 Vgl. http://udn.epicgames.com/Three/TerrainDesign.html#Terrain Design: Guidelines and Information (Entnahmedatum: 02.01.2012)

97 http://www.planetside.co.uk/ (Entnahmedatum: 02.01.2012)

98 Vgl. http://udn.epicgames.com/Three/TerrainDesign.html#Terrain Design: Guidelines and Information (Entnahmedatum: 02.01.2012)

rain also beispielsweise 100 x 100 Einheiten groß werden, muss die Height-Map 101 x 101 Pixel groß sein. Außerdem sollte eine Height-Map als 16 Bit Graustufen-Bitmap importiert werden[99]. Die Helligkeit der einzelnen Pixel definiert hierbei die Höhe des Abschnitts im Terrain. Schwarz steht hierbei für den tiefsten Punkt und weiß für den höchsten. Niedrigere Bit-Tiefen als 16 Bit erzeugen kantige unbrauchbare Landschaften, da sie nicht genügend Höheninformation enthalten.

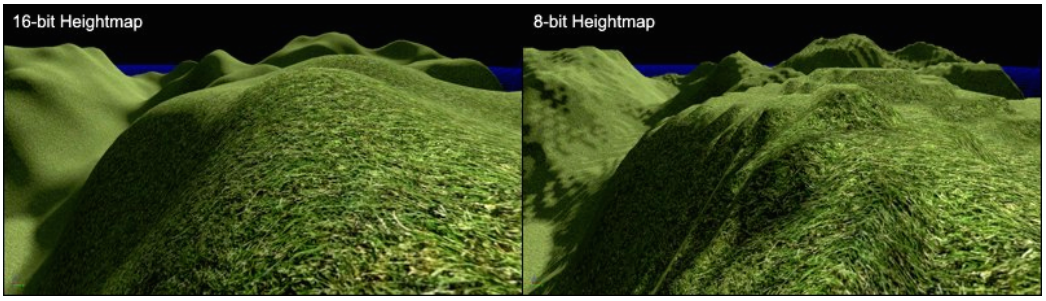

Abbildung 22: Map-Imports mit unterschiedlicher Bit-Tiefe[100]

Vielmehr als der Import von Height Maps empfiehlt es sich, das Terrain direkt im UDK zu erstellen. Es ist hierbei möglich, die Landschaft ähnlich wie bei Sculpting-Programmen wie Mudbox oder Zbrush (vergleiche Kapitel 4.1) zu gestalten. Es lassen sich schroffe Berge, aber auch sanfte Hügel und Täler erstellen. Mittels unterschiedlicher Ebenen lassen sich Materialien, mittels Alphakanal, direkt auf das Terrain malen. Alternativ ist es möglich, einen Alphakanal getrennt zu importieren. Es lassen sich mehrere Material-Ebenen übereinander erstellen. Diese lassen sich auch verschieben, wobei der Alphakanal nicht mit verschoben wird. Das Malen des Materials kann im UDK immer in Verbindung mit dem Terrain geschehen. [101]

Abbildung 23: Terrain mit Materialebenen[102]

99 http://udn.epicgames.com/Three/TerrainDesign.html#Overview (Entnahmedatum: 02.01.2012)

100 http://udn.epicgames.com/Three/TerrainDesign.html#Terrain Design: Guidelines and Information (Entnahmedatum: 02.01.2012)

101 Vgl. http://udn.epicgames.com/Three/TerrainDesign.html#Terrain Design: Guidelines and Information (Entnahmedatum: 02.01.2012)

102 http://udn.epicgames.com/Three/DevelopmentKitBuildUpgradeNotes.html (Entnahmedatum: 02.01.2012)

6.3 Erstellung von Vegetation

Die Vegetation wird im UDK als Foliage bezeichnet. Nachdem im Foliage System Objekte für die Vegetation gewählt wurden, können diese mit einem Pinsel auf das Terrain gemalt werden. Für jede Pflanze lässt sich eigenständig Vegetationsdichte, die Größenvariation, die Neigung und die Vegetationsgrenze anpassen. Es stellte sich heraus, dass es ausreicht zwei bis drei unterschiedliche Bäume sowie Büsche und Farne in unterschiedlicher Höhe zu erstellen. Dies genügt, um eine abwechslungsreiche Vegetation zu schaffen. Hierbei ist zu beachten, dass die Polygonanzahl der Objekte, die für die Vegetation genutzt werden, eingeschränkt sein sollte. Nur so lässt sich eine schnelle und fehlerfreie Berechnung der Bilder garantieren.[103]

Abbildung 24: Anwendung von Foliage im UDK (Eigene Darstellung)

Abbildung 24 zeigt, dass es sich nicht empfiehlt für Nahaufnahmen das Foliage-Tool zu verwenden, da die Erstellung der Vegetation mit dem Foliage-Pinsel zu zufällig ist. Vegetation, die von Nahem betrachtet wird oder auch als Setting dienen soll, sollte von Hand gesetzt und erstellt werden.

6.4 Urbane Szenerien

Um Rechenzeit einzusparen, sollten urbane Szenerien in erster Linie mit BSP[104]-Pinseln realisiert werden. Aus diesen werden CSGs[105] berechnet, die optimiert dargestellt werden können. Gebäude sollten grundsätzlich aus BSP-Pinseln erstellt werden. Sie

103 Vgl. http://udn.epicgames.com/Three/Foliage.html (Entnahmedatum: 02.01.2012)

104 BSP: Definiert, je nach Typ ein erstelltes Volumen (z.B. additiv, subrtraktiv)

105 CSG: UDK-eigene Geometrie, aus Grundkörpern die sich mit Boolschen Operationen beeinflussen lassen

können auch mit Materialien versehen werden. Mit Booleschen Operationen (also Addition, Subtraktion, Schnittmenge) können komplexere Formen erstellt werden.[106] Eine CSG entsteht immer aus der niedrigst möglichen Menge an Polygonen. Die Grundstrukturen von einem Gebäude oder einem Straßenzug können so realisiert werden. Für Türen, Fenster oder andere Verzierungen sollten die Objekte an die CSG gelegt werden. Hiermit lassen sich Rechenzeit und Optik optimieren. Für die Erstellung der Materialien sollten kachelbare Materialien gewählt werden, damit die höchstmögliche Detail-Auflösung gewährt werden kann. Um Verzierungen oder nicht wiederholende Elemente darzustellen, also zum Beispiel Graffiti oder Verschmutzungen an den Wänden, sollten Decals erstellt werden. Dies sind Materialien mit Alphakanal, die auf ein anderes Material platziert werden können. Gebäude sollten keinesfalls aus einem Objekt bestehen, da für so große Objekte keine Möglichkeit besteht, eine funktionierende Lichtberechnung zu realisieren. Des Weiteren kann das Gebäude so nicht flexibel auf das benötigte Umfeld angepasst werden. Wenn kein BSP-Pinsel verwendet werden kann empfiehlt es sich, kachelbare Module zu erstellen, die direkt im UDK platziert werden können.

6.5 Import von Objekten in das UDK

6.5.1 Allgemeines

Szenenobjekte und Charaktere werden beim Import in das UDK in Paketen organisiert. Diese Pakete können einzeln hinzugeladen werden und beinhalten 3D-Modelle, Texturen, Materialien, Partikelsysteme und weitere UDK-spezifische Funktionen.

Der Editor des UDK besteht aus mehreren Unterprogrammen. So besteht die Möglichkeit mit dem Werkzeug Speedtree realistische Bäume zu generieren.[107] Mittels dieser Funktionalität lassen sich schnell variierende Landschaften generieren. Weitere Programme sind FaceFX, der Physical-Asset-Editor und der Partikelsystem-Editor Cascade. Viele Unterprogramme des UDK basieren auf Nodes. Diese verbinden mit Eigenschaften versehene Elemente, die sich miteinander kombinieren lassen. Die Verwendung von Nodes erlaubt eine intuitive Arbeitsweise, da so schneller ersichtlich wird, wie ein Material oder ein ausgelöstes Ereignis aufgebaut ist.

106 Vgl. http://udn.epicgames.com/Three/UsingBspBrushes.html (Entnahmedatum: 02.01.2012)

107 Vgl. http://www.speedtree.com/ (Entnahmedatum: 09.01.2012)

Bei der Erstellung der Testszenen ließ sich feststellen, dass es sich empfiehlt Szenenobjekte und Charaktere beim Import direkt in Gruppen zu ordnen und zu kategorisieren. So bleibt innerhalb eines Pakets stets alles geordnet. Beispielsweise empfiehlt sich ein Unterordner jeweils für Models, Texturen und Materialien. Alle weiteren Objekte, wie zum Beispiel AnimSets, können auch im Hauptordner liegen. Es ließ sich feststellen, dass sich durch die Verwendung von Unterordnern die Pakete bereits ohne Anwendung von Filteroptionen gezielt nach Texturen oder 3D-Modellen durchsuchen lassen.

6.5.2 Charaktere und Objekte

Der Import von 3D-Modellen erfolgt in zwei Gruppen: die StaticMeshes sowie die SkeletalMeshes. Als StaticMeshes werden alle Modelle ohne Bones bezeichnet. Sobald Bones im Modell vorhanden sind, werden die Objekte als SkeletalMeshes importiert. Bei StaticMeshes ist es, falls sich mehrere Objekte in einer Datei befinden, zudem wichtig, ob diese kombiniert oder einzeln importiert werden sollen. Beim Kombinieren werden alle in der Szene befindlichen 3D-Modelle zu einem Objekt verschmolzen und auch im UDK als ein einziges angesehen. Ohne Kombination der Objekte werden diese als einzelne Objekte in das UDK importiert.[108]

6.5.3 Texturen

Der Import von Texturen gleicht dem von 3D-Modellen. Diese werden beim Import komprimiert, um den Speicherbedarf zu minimieren. Bei Texturen ist es wichtig, diese in spezielle Detailgruppen zu unterteilen. Für Normal Maps gibt es beispielsweise TC_Normalmap und TC_NormalmapUncompressed.

- TC_Normalmap besitzt eine DXT1Kompression, die geringen Speicherbedarf zur Folge hat. Nachteil dieser Methode können grob wirkende Normal Maps sein.
- TC_NormalmapUncompressed liefert die besten optischen Ergebnisse. Die Auflösung wird bei der Auswahl dieser Detailgruppe automatisch um die Hälfte verringert. Hierbei hat sie immer noch eine bessere Qualität als TC_Normalmap.[109]

108 Vgl. http://udn.epicgames.com/Three/FBXImporterUserGuide.html (Entnahmedatum: 02.01.2012)

109 http://udn.epicgames.com/Three/NormalMapFormats.html#TC_NormalmapUncompressed / V8U8 (Entnahmedatum: 02.01.2012)

Die Texturgruppe von Normal Maps sollte World Normal Map sein. Ohne diese Einstellungen wirken Normal Maps verwaschen und kontrastarm. Weiteres dazu lässt sich im Unreal Developer Network unter www.udn.epicgames.com (Stand 02.01.2012) nachschlagen.

Anstatt Texturen doppelt zu importieren, wenn sie von mehreren Objekten verwendet werden, empfiehlt es sich, diese aus anderen Paketen zu verwenden. Dies spart Speicherplatz und verringert die Ladezeiten, da die Textur somit nur einmal geladen werden muss. Jedoch funktioniert die referenzierte Textur nur dann, wenn auch das Paket, auf welches zugegriffen wird, vorhanden ist. Dies sorgt wiederum für einen höheren Speicherbedarf. Texturen werden im UDK mit unterschiedlichen Verfahren komprimiert, um von der Grafikkarte optimal verarbeitet werden zu können. Das DXT1-Verfahren hat ein Kompressionsverhältnis von 8:1, während DXT5 mit 4:1 komprimiert. Bei beiden Verfahren handelt es sich um verlustbehaftete Kompression.[110]

Auflösung	**DXT1**	**DXT5**	**unkomprimiert 32Bit**
16x16px	312 Byte	496 byte	4kB
32x32px	824 Byte	1,48kB	8kB
128x128px	10,8kB	21,4kB	52kB
256x256px	42,8kB	85,4kB	196kB
512x512px	170kB	341kB	768kB
1024x1024px	682kB	1.33MB	3.00MB
2048x2048px	2,66MB	5,33MB	12.00MB

Tabelle 2: Typische Texturauflösungen sowie deren Speicherbedarf verglichen mit unkomprimierter 32Bit TGA.

6.6 Weiterverarbeitung des Contents

6.6.1 Materialien

Materialien bestimmen die Oberflächeneigenschaften von Objekten. Sie definieren das Aussehen, die Reflektivität oder beispielsweise das Verhalten des Glanzpunktes eines

110 http://udn.epicgames.com/Three/TextureSupportAndSettings.html#Compressed Texture Memory Requirements (Entnahmedatum: 02.01.2012)

3D-Modells. Der Materialeditor des UDK ist ein nodebasiertes System. Es ist möglich, verschiedene Textur-, Parameter- und Operationsnodes miteinander zu verbinden. Alle Nodes laufen in einer Main Node zusammen. Diese definiert das finale Material und kombiniert alle dahinter befindlichen Nodes. Die wichtigsten Eingänge dieser Node, die für die Erstellung eines Materials benötigt werden, sind die Diffuse-, Specular- und Normal Node. Weitere Werte ermöglichen zusätzliche Effekte, wie unter anderem selbstleuchtende oder transparente Materialien. Für einfache Materialien genügt es, die Diffuse-, Specular- und Normal Map mit der Main Node zu verbinden. Um mehr Kontrolle über das Material zu haben, ist es zudem möglich, einzelne Texturen im Materialeditor über Nodes zu kombinieren. Somit ist es beispielsweise möglich, mehrere Texturen ineinander zu multiplizieren oder eine Normal Map mit einer anderen zu überlagern. Für die realistische Hautdarstellung von Charakteren unterstützt die Unreal Engine 3 Subsurface Scattering, kurz SSS.[111] Diese Methode erlaubt die Berechnung der Lichtstreuung in lichtdurchlässigen Objekten. Es ist dadurch möglich, das auf die Haut einfallende Licht teilweise durchzulassen. Sichtbar ist dies beispielsweise bei einer Hand, die vor eine eingeschaltete Glühbirne gehalten wird. Um mehrere Varianten eines Materials zu erstellen, gibt es Material-Instanzen. Mit diesen lassen sich Parameter der zugehörigen Hauptmaterialien ändern, ohne diese selbst zu beeinflussen.[112] Hierzu müssen einzelne Konstanten und Texturen des Hauptmaterials zu Parametern konvertiert werden. Abbildung 25 zeigt die über Material-Instanzen geänderte Farbe an zwei identischen Objekten.

Abbildung 25: wechselbare Diffuse Color einer Materialinstanz (Eigene Darstellung)

111 Vgl. http://udn.epicgames.com/Three/ScreenSpaceSubsurfaceScattering.html (Entnahmedatum: 02.01.2012)

112 Vgl. http://udn.epicgames.com/Three/MaterialInstanceConstant.html#Overview (Entnahmedatum: 02.01.2012)

6.6.2 Echtzeitreflexionen

Bei der Produktion von herkömmlichen 3D-Animationsfilmen können über Raytracing[113] realistische Reflexionen erzeugt werden. In der Echtzeitanwendung ist dies nicht möglich, da die in der Offline-Produktion verwendeten Algorithmen sehr rechenintensiv sind. Deshalb werden in Games Engines sogenannte Cube Maps verwendet. Dies sind Texturen, die ein 360°-Abbild einer Umgebung darstellen. Innerhalb des UDK lassen sich diese Cube Maps über den Cubemap-Actor erstellen. Dieser erstellt ein Abbild der Umgebung ausgehend von seiner Position. Diese Umgebungstextur lässt sich innerhalb von Materialien verwenden, um Reflexionen zu simulieren. Ein Problem hierbei ist, dass die Reflexionsberechnung vorab erfolgen muss. Dies hat zur Folge, dass nur statische Objekte innerhalb der Cube Map abgebildet werden können. Charaktere sind deshalb auf einer reflektierenden Oberfläche nicht zu sehen. In der Praxis fällt dies allerdings nur bei stark spiegelnden Objekten auf. Bei Metallflächen, die nur sehr diffuse Reflektionen aufweisen, wird dieses Problem weniger deutlich als bei einem Spiegel. Wenn es dennoch erwünscht ist, dass sich Charaktere in Objekten spiegeln, können Spiegelmaterialien erstellt werden.[114] Dies ist jedoch sehr rechenintensiv. Objekte, die sich weit weg befinden, werden nicht reflektiert. Dies ist aus Performance-Gründen nötig.

6.6.3 Decals

Decals bezeichnen spezielle Materialarten, mit denen es möglich ist, zusätzliches Detail beziehungsweise Variation in Umgebungen zu bringen. Sie werden wie Aufkleber auf bestehende Objekte in einer Szene angewendet. Meist bestehen sie aus mit Alphakanal versehenen Texturen, die es ermöglichen Schilder, Graffiti oder Dreck auf Objekten darzustellen. Bei der Erstellung der Testszenen wurde deutlich, dass dies nötig ist, um Details in kachelbare Umgebungen einzufügen. Innerhalb des UDK besitzen Decals ein Volumen. Dieses beschreibt, in welchem Bereich das Decal angewandt wird. Alle innerhalb dieses Volumens befindlichen Objekte bekommen das Decal zugewiesen.[115]

---

113 Handbook of Multimedia for Digital Entertainment and Arts, Seite 530

114 http://udn.epicgames.com/Three/RenderToTexture.html#SceneCaptureReflectActor (Dynamic Reflections) (Entnahmedatum: 02.01.2012)

115 http://udn.epicgames.com/Three/UsingDecals.html#What are decals? (Entnahmedatum: 02.01.2012)

6.6.4 Physiksimulation

Für die automatische Physiksimulation innerhalb der Unreal sind Physical Assets nötig. Physical Assets dienen dazu die physikalischen Eigenschaften von Objekten zu definieren. Diese lassen sich entweder durch Spieleraktionen oder geskriptete Ereignisse auslösen.[116] Komplexe Animationen fallender Objekte oder Charaktere lassen sich somit automatisch animieren. Die Erstellung der physikalischen Eigenschaften von Objekten erfolgt mit dem Physical-Asset-Tool.[117] Auswirkungen von Interaktionen auf statische Objekte, wie zum Beispiel ein gegen eine Wand fahrendes Auto, lassen sich mittels UDK ebenfalls automatisch berechnen. Hierzu dient das Fracture Tool. Damit lassen sich Bruchstellen innerhalb von Objekten definieren. Interagiert eine Figur mit diesem Objekt und beschädigt es dabei mit der benötigten Stärke, zerbricht es.[118]

6.6.5 Lichtberechnung

Zur korrekten Darstellung von statischen Objekten in der Unreal Engine muss die Beleuchtung vorberechnet werden. Dies erfolgt über ein Unterprogramm mit Namen Lightmass.[119] Hierbei handelt es sich um die Simulation globaler Beleuchtung. Diese ermittelt die Verteilung der Lichtphotonen im Raum und schafft somit die statische Beleuchtung. Durch das Abprallen des Lichtes werden auch eigentlich im Schatten liegende Objekte leicht beleuchtet. Hierdurch werden Szenen, die Im Editor meist sehr dunkel erscheinen, noch einmal aufgehellt.[120]

Beim Erstellen der Testszenen ergab sich, dass zur korrekten Berechnung des Lichtes eine korrekt eingestellte und gewählte UVW Map sowie Lightmap-Auflösung benötigt wird. Diese müssen für statische Objekte sowie BSP-Strukturen definiert werden.[121] Eine hohe Lightmap-Auflösung eines statischen Objektes sorgt in Verbindung mit einer niedrigen Auflösung auf der darunterliegenden BSP-Struktur dafür, dass die vorberechneten Schatten auf dem statischen Objekt scharf sind, während die BSP-Struktur über sehr unscharfe Schatten verfügt. Dadurch lässt sich feststellen, dass die Dichte der

116 Vgl. Mastering Unreal Technology, Volume 1, Seite 16

117 http://udn.epicgames.com/Three/PhATUserGuide.html (Entnahmedatum: 02.01.2012)

118 Vgl. http://udn.epicgames.com/Three/FractureTool.html (Entnahmedatum: 02.01.2012)

119 Vgl. http://udn.epicgames.com/Three/Lightmass.html (Entnahmedatum: 02.01.2012)

120 Vgl. http://udn.epicgames.com/Three/Lightmass.html#Lightmass Static Global Illumination (Entnahmedatum: 02.01.2012)

121 Vgl. http://udn.epicgames.com/Three/ShadowingReference.html#Precomputed Shadows and BSP (Entnahmedatum: 02.01.2012)

Lightmaps möglichst gleichmäßig gewählt werden sollte, da die Beleuchtung sonst ungleichmäßig detailliert erscheint. Eine Beschleunigung der Lichtberechnung ist mittels Lightmass Importance Volumes (deutsch etwa: „Lightmass“ Relevanzvolumen) möglich. Diese definieren den Bereich, in dem der Lichtberechnung eine höhere Priorität zugeteilt wird. Objekte innerhalb des Volumens bekommen eine detailliertere Beleuchtung. Außerhalb liegende Objekte erhalten nur eine Berechnung des indirekten Lichtes in niedriger Qualität.[122] Mittels Lightmass können realistisch wirkende weiche Schatten erzeugt werden, wie sie bei der Verwendung von großen Lichtquellen auftreten. Je größer die Lichtquelle, desto weicher verhalten sich die Schatten.

6.6.6 Optimierung der Echtzeitfähigkeit

Die Echtzeitdarstellung im UDK lässt sich auf mehrfache Weise erhöhen. Dies ist vor allem nützlich, wenn es um die Vorvisualisierung eines Films in niedrigerer Qualität geht. Eine Möglichkeit ist der Verzicht auf komplex modellierte Umgebungen. Hierbei werden nur die für die Szene nötigen Models verwendet. Ein Mittel, das festgestellt werden konnte, ist eine Umgebungstextur, die ein 360°-Abbild der Szene enthält. So lassen sich nur für die Szene nötige Models verwenden. Dadurch lässt sich beispielsweise ein komplexer Wald, welcher sich nicht in Echtzeit darstellen lässt, in eine Textur backen. Dies ermöglicht es, eine Szene verlustfrei zu betrachten, da das UDK bei einer stark ruckelnden Echtzeitdarstellung Einzelbilder überspringt. Eine weitere Optimierungsmöglichkeit ist die Verwendung von Level of Detail 3D-Models. Level of Detail bezeichnet eine weniger detaillierte Versionen der Modelle, diese werden in Abhängigkeit der Distanz zum Betrachter verwendet. Diese Varianten des Models lassen sich im jeweiligen 3D-Bearbeitungsprogramm erstellen und in das UDK exportieren. Eine andere Möglichkeit ist die Verwendung des UDK-internen Vereinfachungsalgorithmus. Mit dem Simplygon-Werkzeug[123] lassen sich Level of Detail Models innerhalb des UDK-Editors generieren. Hierbei hat der Künstler allerdings weniger Kontrolle über das Endergebnis.

122 Vgl. http://udn.epicgames.com/Three/Lightmass.html#LightmassImportanceVolume (Entnahmedatum: 02.01.2012)

123 http://udn.epicgames.com/Three/MeshSimplificationTool.html Entnahmedatum: 02.01.2012

7 Anwendung des Gamecast-Systems

7.1 Allgemeines

Gamecast ist ein virtuelles Kinematographie-System, das an der Hochschule Mittweida entwickelt wurde. Mit dem System ist es möglich, innerhalb des laufenden Spiels die Aktionen der Spieler sowie deren Emotionen aufzuzeichnen. Ziel ist es, die aufgezeichneten Daten wiederzuverwenden, um einen Film zu generieren. Hierbei soll es möglich sein, die Daten nach dem laufenden Spiel zu analysieren und bearbeiten. Hierfür wurden an der Hochschule Mittweida Werkzeuge wie eine Emotionsunterstützung für das UDK, ein Loggingtool sowie ein Programm zum Auswerten der Log-Daten entwickelt. Ein Loggingtool ist ein Programm zur Aufzeichnung von Positions- und Emotionsdaten, den sogenannten Log Daten. Im Rahmen der Erstellung des animierten Films für diese Bachelorarbeit wurde das Gamecast-System einem Praxistest unterzogen und für die Aufzeichnung zweier Szenen verwendet. Ziel dieses Tests ist es, Möglichkeiten sowie Probleme des Systems aufzuzeigen, um zu einer Verbesserung beizutragen.

7.2 Vorbereitung von Charakteren

Für die Verwendung der zur Verfügung stehenden Charaktere sind funktionierende AnimTrees notwendig. Diese müssen darauf ausgelegt sein, im Spiel zu funktionieren und alle Möglichkeiten des Spielers abzudecken. Einfache AnimTrees, die nur für die Verwendung in Unreal Matinee konzipiert sind, funktionieren hier nicht, da sich lediglich die Emotionen ansteuern lassen. Zur Verwendung bereits bestehender AnimTrees sind kompatible Benennungen der Bones nötig. Dies ermöglicht es, im UDK vorhandene AnimTrees zu übernehmen. Das Virtual Actor[124] genannte Emotionserkennungssystem erlaubt es, über eine Webcam die Mimik des Spielers zu erfassen und auf seinen virtuellen Avatar zu Übertragen. Für die Verwendung sind spezielle Morphnode Namen innerhalb des AnimTrees nötig.

Das Gamecast-System erlaubt die Verwendung von Morphtargets zur Emotionsaufzeichnung die in der folgenden Tabelle 3 zusammengefasst sind:

124 http://www.gamecast-tv.com/projekt/virtual-actor/ (Entnahmedatum: 03.01.2012)

Gesichtsausdruck	UDK Morphnode
Wütend	emote_angry
Traurig	emote_sad
Glücklich	emote_smile
Überrascht	emote_surprised
Mund geöffnet	emote_mouthopen
Linkes Auge schließen	emote_blinkleft
Rechtes Auge schließen	emote_blinkright

Tabelle 3: Gamecast Morphnodes für die Emotionserkennung.

Zusätzliche Animationen, welche als Taunts bezeichnet werden, müssen in eine Textdatei eingetragen werden, beispielsweise ein Hammerschlag auf einen Nagel. Diese werden über den Nummernblock ausgeführt. Es ist hierbei möglich, bis zu neun verschiedene Animationen per Tastendruck auszulösen. Falls mehr Animationen benötigt werden, müssen diese in einem zweiten Aufzeichnungsdurchlauf mittels Gamecast aufgezeichnet werden. Hierbei muss vorher die Textdatei mit den Verweisen auf die Taunts erneut bearbeitet werden.

7.3 Produktionsablauf im Gamecast-System

Gesteuert wird das Gamecast-System von einem Manager-Prozess. Mit diesem Manager verbinden sich alle aktiven Server. Der Logging-Prozess läuft über diese Anwendung. Für das Ausführen des Systems ist eine Stapelverarbeitungsdatei verantwortlich. Diese startet einen Server mit dem gewünschten Level. Dieser Server verbindet sich automatisch zum Manager-Prozess. Voraussetzung hierfür ist, dass dieser Level mit korrektem Pfad in Relation zum UDK-Installationspfad angegeben ist.

Mit einer zweiten Stapelverarbeitungsdatei verbindet sich der Spieler mit dem Server. In dieser wird ebenfalls festgelegt, welches Aussehen der Spieler während der Aufzeichnung besitzt. Je nachdem, wie viele Spieler dem Server beitreten müssen, muss dieser Prozess wiederholt werden. Bei Charakteren, die über die gleichen AnimTrees verfügen, ist es irrelevant, welches Aussehen die Spieler besitzen, da dies im Nachhinein geändert werden kann.

Die Aufzeichnung der Spielszenen mittels Gamecast startet, sobald ein Spieler den Server betreten hat und sich in der Szene befindet. Während der Aufzeichnung werden

alle von den Spielern durchgeführten Handlungen aufgezeichnet. Es ist also wichtig, sich bereits vor dem Aufzeichnungsstart im Klaren zu sein, welche Szene gespielt werden soll. Nach Aufzeichnungsende müssen alle Spieler den Server verlassen und das Serverprogramm muss beendet werden, da sonst die Aufzeichnung weiterläuft. Der Server lässt sich derzeit nur beenden, indem das Kommandozeilenfenster des Servers geschlossen wird. Über den Logfilekonverter können die aufgezeichneten Daten in Unreal Kismet als Matinee-Daten importiert werden. Der Konverter besitzt hierfür eine Exportfunktion der mit Gamecast aufgezeichneten Daten. Diese werden in das von Gamecast entwickelte GCL-Format konvertiert. Vor dem Export ist es möglich, die später benötigte Datendichte zu wählen. Die Datendichte definiert, wie oft in einem bestimmten Zeitraum Daten erfasst werden. Der Export erfolgt automatisch als Matinee-Data in Unreal Kismet. Um weitere Aufzeichnungen in das UDK zu laden, müssen diese Gamecast Logfiles mit einem Texteditor geöffnet werden. Die Inhalte der Dateien werden vollständig in die Zwischenablage kopiert und in Unreal Kismet eingefügt. Die Matinee-Daten lassen sich mit beliebigen Charakteren verbinden. Hierbei ist zu beachten, dass die Zuordnung der Matinee-Charaktere in der Reihenfolge erstellt wird, in der die Spieler den Server betreten haben.

7.4 Praxisanalyse des Gamecast-Systems

7.4.1 Verwertung von Movement Keyframes

Der Import ins Matinee sorgt bei Standardeinstellungen des Konverters für eine hohe Anzahl von Keyframes. Grund hierfür ist die hohe Keyframe-Anzahl, die bei der Aufzeichnung entsteht. Die Keyframes speichern die Positions- und Emotionsdaten der Spielfiguren im dreidimensionalen Raum. Diese lassen sich nur schwer bearbeiten, denn Unreal Matinee erlaubt es zwar, mehrere Keyframes innerhalb der Zeitleiste zu verschieben. Allerdings ist es nicht möglich, diese Keyframes im dreidimensionalen Raum als Gruppe zu bewegen. Unnötige Keyframes, die beim Spielstart entstehen, lassen sich bereits während des Exports zu Matinee beseitigen. Diese Keyframes können beispielsweise das Positionieren der Spieler in der Szene beinhalten. Es ist möglich, diese Keyframes über eine Änderung der Exportstartzeit zu entfernen.

Der Import der Gamecast-Daten mit geringeren Keyframeraten in das UDK sorgt bei der aktuellen Version des Gamecast Logfilekonverters für eine simple Reduktion der Keyframes. Hierbei werden Keyframes ohne Beachtung ihrer Relevanz herausge-

löscht. Dies resultiert besonders bei geringen Keyframeraten in schwammigen Bewegungen. Im Vergleich zur hohen Datendichte der Aufzeichnung scheint ein Charakter beispielsweise bei einer 3-prozentigen Genauigkeit über den Boden zu schwimmen, obwohl er sich bei 100 Prozent nicht bewegt. Anlage 1 zeigt Datenabweichungen in Abhängig der Exportdichte. Die Koordinaten definieren die relative Abweichung der Koordinaten in UU von 100% Datendichte. Hierbei lässt sich die Schlussfolgerung ziehen, dass nur 100% und 50% Datendichte nutzbar sind. Bei niedrigeren Datendichten wird die Abweichung der Position zu groß. Dieses simple Löschen von Keyframes hat zur Folge, dass entweder die volle Dichte an Keyframes importiert werden muss. Dabei müssen irrelevante Keyframes von Hand gelöscht werden. Alternativ lassen sich geringe Datendichten importieren und manuell anpassen. Der Arbeitsaufwand bei beiden Prozesse gleicht sich, da jeweils ein hoher Optimierungsbedarf besteht.

Die Keyframes des aktuellen Konverters werden als Worldframe Keyframes mit Bezier-Eigenschaften importiert. Dies lässt die Schlussfolgerung zu, dass es nicht möglich ist, über die Position des Charakters im Editor den Start der Animationen festzulegen. Durch die Verwendung relativer Keyframes ist dies möglich, da diese sich immer am Ursprung ausrichten. Der Ursprung ist im Matinee der Punkt, an dem der Charakter in der Szene steht. Ein weiterer Vorteil relativer Keyframes ist die Möglichkeit, einen Animationsabschnitt mehrfach unabhängig von der aktuellen Positionierung zu verwenden. Bei der erneuten Verwendung eines World Keyframes springt der Charakter zur Position, an der der World Keyframe erstellt wurde. Die Kurveneigenschaften der Bezier Keyframes wirken sich bei der Standard-Datendichte nicht negativ aus. Bei geringen Keyframe-Raten sorgen sie allerdings für eine Beschleunigung sowie ein Abbremsen der Animationen. Dies lässt sich beheben, indem die Keyframes im UDK manuell zu Linearen umgewandelt werden. Bei diesen ist die Geschwindigkeit immer konstant.

7.4.2 Anwendung der Emotionsdaten

Für den Erhalt brauchbarer Emotionsdaten ist eine gute Ausleuchtung des virtuellen Schauspielers während der Aufnahme nötig. Das Programm, das die Kamera anspricht, muss in der Lage sein, das komplette Gesicht des Nutzers zu analysieren. Haare, die die Augen oder Augenbrauen des Nutzers verdecken, wirken sich negativ auf die Aufnahme aus. Nutzer mit Brille werden allerdings nicht dazu gezwungen, ihre Brille abzulegen. Die Datendichte der Emotionskeyframes ist ebenfalls sehr hoch. Diese entsteht durch das kontinuierliche Abtasten des Gesichts des Spielers. Die Emotionserkennung ist außerdem noch ungenau. Die Aufnahme geschlossener Augen resul-

tiert teilweise in einem kontinuierlich ausgeführten Zwinkern anstatt einem Schließen der Augen. Die Änderungen der Emotionen erfolgen stellenweise auch zu schnell. Es kann vorkommen, dass zwischen zwei Keyframes plötzlich der Zustand von glücklich auf wütend wechselt. Ein manuelles Bearbeiten und Ausdünnen der Emotionsdaten ist hier nötig. Derzeit ist es noch nötig, die Emotionserkennung auf jeden Protagonisten zu kalibrieren. Dies erlaubt keinen schnellen Benutzerwechsel, da ansonsten Emotionen falsch gedeutet werden können. Beispielsweise kann es vorkommen, dass das Programm bei einem neutralen Gesichtsausdruck traurige Emotionen interpretiert. Die Emotionen müssen vom Spieler stärker als im realen Leben ausgedrückt werden, um in der Games Engine sichtbar zu werden. Hierdurch wird das Arbeiten mit den Emotionen ermüdend, da man sich stark auf den eigenen Gesichtsausdruck konzentrieren muss. Aufgrund der Verarbeitungsdauer durch die Emotionserkennungssoftware ist die Echtzeittauglichkeit nicht gewährleistet. Bei der Aufzeichnung einer einzelnen Figur ist dies nicht störend. Problematisch wird dies bei mehreren virtuellen Schauspielern. Diese sehen die emotionalen Reaktionen der anderen Charaktere teilweise eine halbe bis eine Sekunde verzögert.Sprechen während der Aufnahme wird unter Umständen als emotionaler Ausdruck gedeutet. Dies hat zur Folge, dass Phoneme wie ein „O“ als überraschter Gesichtsausdruck gewertet werden.

7.4.3 Iteratives Aufzeichnen

Iteratives Aufzeichnen ist das schrittweise Wiederholen von Aufzeichnungsvorgängen um sich den exakten Daten anzunähern. Bei einer Szene mit zwei Akteuren kann ein Akteur aufgezeichnet werden. Diese Daten können während der Aufzeichnung des zweiten Akteurs abgespielt werden. So hat der Benutzer einen Anhaltspunkt für seine Aktionen. Dieser Prozess ist allerdings noch umständlich. Hierbei muss eine Sequenz aufgenommen und exportiert werden. Die exportierte Sequenz lässt sich in Unreal Matinee in einem erneuten Aufzeichnungsdurchlauf abspielen. Während diese Sequenz abgespielt wird, ist es möglich eine neue Aufnahme auszuführen, um somit einen zweiten Charakter oder die Emotionen für den ersten Charakter aufzuzeichnen. Durch Kombination beider Matinee-Sequenzen lassen sich so iterativ Daten aufzeichnen. Das Einfügen der Daten aus einer zweiten Matinee-Sequenz ist derzeit nur innerhalb von Unreal Matinee und Kismet möglich. Hierzu müssen die aufgenommenen Spuren der Quellsequenz kopiert und in die Zielsequenz eingefügt werden. Eine Kombination von unterschiedlichen aufgenommenen Spuren und Charakteren ist möglich. Somit können beispielsweise nach einer Aufnahme die Emotionsspuren zweier Charaktere getauscht werden.

7.5 Optimierungsansätze für Gamecast

In diesem Abschnitt werden Ansätze aufgezeigt, die sich während des Praxistests als problematisch oder umständlich erwiesen haben. Weiterhin werden neue Ansätze für die Workflowoptimierung dargelegt.

7.5.1 Einstellung von Sonderanimationen

Die derzeitige Einstellung von Sonderanimationen, den Taunts, ist nur über eine spezielle Textdatei möglich. Fehlerhaft benannte Animationen oder unbeabsichtigte Änderungen durch den Nutzer sorgen für Fehler in der Ausführung. Hierbei bietet es sich an, diese Datei durch eine Nutzeroberfläche zu visualisieren. Über diese GUI könnte der Nummernblock, welcher die Animationen ansteuert, mit Animationen versehen werden. Die Begrenzung der Sonderanimationen auf derzeit neun Sonderanimationen ließe sich durch eine Doppelbelegung des Nummernblocks erhöhen. Um dies zu ermöglichen, muss eine Auswahl der Belegungsebenen durch Tastenbefehle möglich sein. Beispielsweise wäre Animationsebene 1 durch die Taste F1 definiert, während F2 die Folgeebene darstellt. Diese Sonderanimationen könnten sich auch für mehr als nur Emotionen nutzen lassen. Es wäre möglich, gezielt Ereignisse auf Tastendruck auszulösen. Hierbei muss der virtuelle Schauspieler sich nicht auf vordefinierte Skriptereignisse verlassen und womöglich exakt den automatisch ausgelösten Zeitpunkt abwarten, sondern kann diesen selbst bestimmen. Beispielsweise würde eine Mauer bei herkömmlichen Skripts automatisch einstürzen, sobald der Charakter sich dieser nähert. Mittels manuell ausgeführter Umweltereignisse kann der virtuelle Schauspieler also selbst bestimmen, wann ein Ereignis stattfindet. Dies erlaubt eine höhere Kontrolle über das Geschehen am virtuellen Set.

7.5.2 Logfile-Konverter

Die Arbeit mit dem Konverter ermöglicht es derzeit lediglich, die aufgezeichnete Datei zu öffnen und mit verschiedenen Genauigkeitsstufen in das UDK zu importieren.

Eine intelligente Keyframereduktion würde eine Vereinfachung der Nachbearbeitung für den Animator bringen. Diese Keyframereduktion könnte wie im folgenden beschrieben aufgebaut werden. Keyframes werden nicht kontinuierlich gespeichert, sondern orientieren sich an den abgespielten Animationszyklen. Dies hätte zur Folge, dass die Keyframes einer aufgezeichneten Laufbewegung immer exakt nach einem Schritt er-

stellt werden. Da iteratives Aufzeichnen derzeit noch sehr umständlich über das Austauschen von Matinee-Spuren läuft, bietet es sich an, diese Funktionalität in den Konverter zu verlagern. Es sollte somit möglich sein, mehrere einzelne Log-Dateien in eine weiterentwickelte Konverterversion zu laden. Diese Dateien müssten sich untereinander beliebig kombinieren lassen. Dadurch wäre es ohne Umwege möglich, beliebig viele Logfiles miteinander zu kombinieren.

7.5.3 Renderwarteschlange

Die Renderingfunktion des UDK ist zu unflexibel. Sie bietet keine Möglichkeit der Automatisierung des Rendervorgangs. Es sind komplizierte Einstellungen über eine Verknüpfung nötig. Mit Variablen müssen verschiedene Parameter festgelegt werden, die das Rendering beeinflussen. Um dies zu optimieren, bietet sich eine Renderwarteschlange an, die wie folgt aufgebaut sein könnte:

Zentraler Menübestandteil ist ein Listenmenü, in das sich zu rendernde Szenendateien einfügen lassen, sowie eine Wiederholungsfunktion, um einzelne Szenen mehrfach berechnen zu lassen. Dies bietet sich vor allem für Szenen mit Kleidungs- oder Physiksimulationen an, da diese zu unberechenbaren Ergebnissen führen können. Das automatisierte mehrfache Rendering dieser Szenen sorgt für mehr Flexibilität in der Postproduktion. Es können die besten Szenen ausgewählt oder kombiniert werden. Zurzeit muss der Screenshot-Ordner nach jedem Durchlauf manuell geleert werden, da die vorhergehende Sequenz ansonsten überschrieben wird. Bei einer Automatisierung müsste dieses Verschieben in einzelne Unterordner automatisch erfolgen.

8 Matinee

8.1 Allgemeines

Matinee dient zur Erstellung von filmischen Sequenzen in Spielen und um die Eigenschaften von Objekttypen in der Szene mit Keyframes zu steuern[125]. Eine Matinee kann durch verschiedene Ereignisse ausgelöst und abgespielt werden. Dabei wird in Kismet eine Matinee-Node erstellt, die dann mit verschiedenen Nodes angesprochen wird. So ist es möglich, die Matinee direkt von der Spielerstart-Node aus abzuspielen. Um eine Matinee abzuspielen, ist es immer nötig, einen Spielerstart im Level zu platzieren. Dieser Spielerstart ist ein Element, das sich nicht in Matinee steuern lässt. Der Blick erfolgt in der Sequenz nicht wie im Spiel zwangsläufig aus Spieler-Perspektive, sondern durch Kameras. In Matinee gibt es die Möglichkeit, eine Vielzahl von Spuren zu kontrollieren. So ist es möglich, Kameras zu steuern. Hierbei können das Field of View, also das Sichtfeld der Kamera, und die Bewegung kontrolliert werden. Auch die Animationen von Charakteren und ihre Positionsveränderung im Level werden über Matinee gesteuert. Für Morph Targets werden eigene Spuren erstellt. Außerdem ist es möglich, die Kopfdrehung, wenn sie im AnimTree vorgesehen ist, zu steuern.

8.2 Organisation von Matinee-Projektdaten

In Matinee werden Objekte, Charaktere und Kameras in Gruppen verarbeitet. Hierbei lassen sich den Gruppen Register zuweisen. Wenn ein Register ausgewählt ist, werden wie in einer Akte nur die darin befindlichen Gruppen angezeigt. Bei der Erstellung der Testszenen wurde festgestellt, dass es von Vorteil ist, pro Charakter ein Register zu erstellen. Die Gruppen für humanoide Charaktere mit MorphWeights, Animationen, FaceFX und anderem werden sehr groß und die Übersichtlichkeit kann sonst schlecht gewahrt werden. Des Weiteren bietet es sich an, ein Register für die Kameras zu erstellen. Es ist von Vorteil, mit mehreren Kameras zu arbeiten, jeweils eine pro Charakter und eine oder mehrere totale oder freie Kameras. So wird der Schnitt von mehreren Kameras, genannt Multi-Cam-Schnitt, vereinfacht. Dies wurde beim Erstellen der Testszenen deutlich.

125 Vgl. http://udn.epicgames.com/Three/MatineeUserGuide.html (Entnahmedatum: 02.01.2012)

Gruppen lassen sich wiederum in Ordner einteilen. Diese können beliebig ein- oder ausgeblendet werden. Dabei lässt sich zum Beispiel jeweils für einen Charakter und die zugehörige Kamera ein Ordner erstellen. So ist es möglich, nur den Ordner zu öffnen und das Vorschaufenster auf die Kamera festzustellen und so die Spuren für den Charakter zu bearbeiten. Wenn die Kameras so erstellt werden, dass sie immer ein eingerichtetes Bild liefern, wäre es dank des Echtzeit-Renderings möglich, jede Kamera einzeln auszugeben. In der Postproduktion lassen sich mittels Multi-Cam-Schnitt flexibel Bilder schneiden. Dadurch wird der Arbeitsaufwand in der Konzeption verringert. Durch die Vielfältigkeit der Organisation mittels Register und Ordner lässt sich ein guter Arbeitsablauf realisieren. Denn es werden nur Elemente angezeigt, die auch wirklich relevant sind, da alles andere ausgeblendet wird.[126]

8.3 Verwendung von Charakteren in Matinee

Für die Verwendung von Charakteren gilt es in Matinee zu beachten, dass sie für die Verwendung in Matinee ausgewiesen sein müssen. Das heißt, dass sie ein AnimSet und ein AnimTree zugewiesen haben müssen. Außerdem kann ein MorphTargetSet zugewiesen werden. Es ist auch möglich, sie ohne diese Zuweisung zu verwenden. Hierbei ist aber zu beachten, dass es so nicht möglich ist, Animationen zu überblenden, da kein AnimTree zugewiesen werden kann. Des Weiteren können Morph Targets nicht angesprochen werden. Ober- und Unterkörper können nicht mit getrennten Spuren animiert werden und es kann nur ein Anim Control Track verwendet werden. Der Anim Control Track steuert im Matinee, welche Animationen zu bestimmten Zeitpunkten ausgeführt werden. Es lässt sich feststellen, dass Charaktere ohne diese Zuweisungen aus den oben genannten Gründen nur ungenügend für die Erstellung eines Films geeignet sind. Sie dienen eher für die Platzierung eines animierten und Matinee-gesteuerten Charakters in einem Level, das keine komplexen Aktionen ausführen soll.[127]

Es ließ sich feststellen, dass Animationen nur dann im AnimTree verzeichnet werden müssen, wenn in Matinee tatsächlich mit den Nodes überblendet wird. Die Node BlendByIdle ist beispielsweise nur dann nötig, wenn der Charakter bei einer Positionsveränderung im MovementTrack auch tatsächlich laufen soll, ohne dass ihm diese Animation mittels Anim Control Track befohlen wird. Dies ist für die Arbeit im Matinee allerdings

126 Vgl. http://udn.epicgames.com/Three/CreatingCinematics.html (Entnahmedatum: 02.01.2012)

127 Vgl. http://udn.epicgames.com/Three/MatineeAnimControlTrack.html (Entnahmedatum: 02.01.2012)

kaum von Nöten. Jedoch ist die Verzeichnung der Animation im AnimTree dringend für die Benutzung des Charakters im Gamecast-System nötig.

Es gibt zwei Möglichkeiten, Gruppen für die Animation eines Charakters zu erstellen. Gruppen sind in der Ordnungsstruktur von Matinee die Behälter von einzelnen Tracks, also Spuren, auf denen Keyframes mit Zeit und Wert dargestellt werden können. Einerseits lässt sich eine leere Gruppe erstellen, bei der die jeweils benötigten Spuren nachträglich erstellt werden. Andererseits kann auch eine SkeletalMeshGroup erstellt werden, bei der schon ein Movement- und ein Anim Control Track vorhanden sind. Es empfiehlt sich, die Erstellung der leeren Gruppe vorzuziehen. So kann direkt eingestellt werden, ob die Animation nur für den Ober-, Unter- oder den ganzen Körper angewendet werden soll. Abbildung 26 zeigt eine solche SkeletalMeshGroup mit verschiedenen Spuren für die Steuerung eines Charakters.

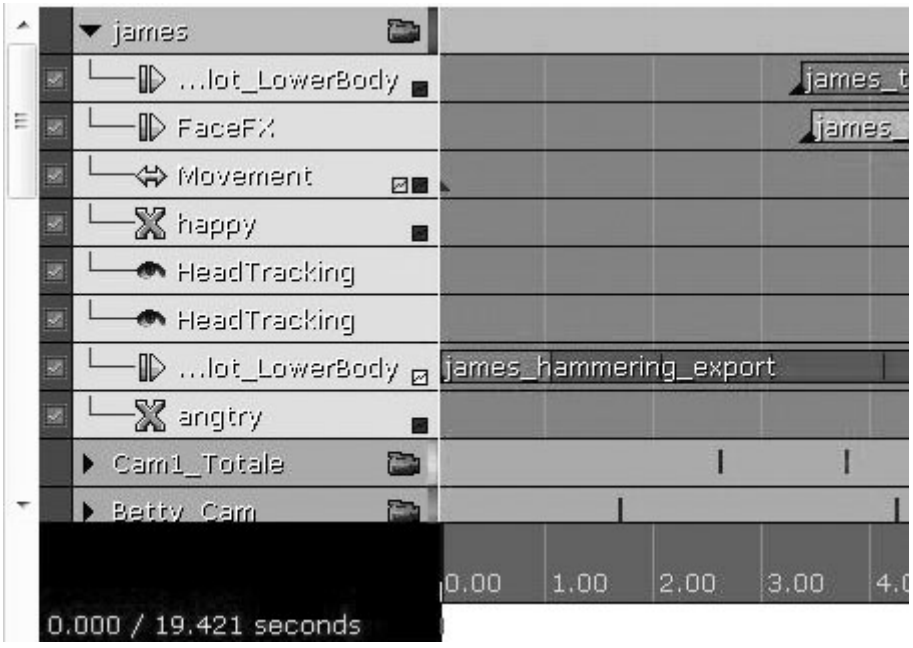

Abbildung 26: SkeletalMeshGroup mit verschiedenen Tracks (Eigene Darstellung)

Wenn der AnimTree funktionsfähig ist und die Körperteile definiert werden, kann zwischen Ober- und Unterkörper überblendet werden und zwei Animationen können gleichzeitig ausgeführt werden. So kann der Charakter beispielsweise mit den Armen kreisen, während er läuft. Hierfür ist es notwendig, Animationen im AnimSet zu überblenden. Diese sind zwar als einzelne Animationssequenzen unbrauchbar, da sie einen Halbwert aus den beiden geblendeten Animationen darstellen, können aber durch Hinzufügen in der entsprechenden Spur für das Überblenden genutzt werden. Weiterhin ist es möglich, zwischen zwei aneinandergrenzenden Animationen zu über-

blenden, indem sie in unterschiedliche Anim Control Tracks gesetzt werden und im Kurveneditor von Matinee ein Crossfade, also eine kreuzende Überblendung erstellt wird.[128] Anim Control Tracks müssen, damit sie vollständig und richtig funktionieren, im Kurveneditor auf einen Wert von 1 gesetzt werden, falls sie nicht im AnimTree verzeichnet sind.

8.3.1 Funktion des Movement Tracks

Mit einem Movement Track lässt sich die Position eines Charakters mittels Keyframes steuern. Hierbei gilt es zu beachten, dass die Keyframes des Charakters voreingestellt relativ sind. Das heißt, sie verwenden keine absoluten Positionen, sondern sind relativ zum Ausgangspunkt. Wenn eine Bewegung mit drei Keyframes realisiert wird und der mittlere Keyframe gelöscht wird, wird der dritte Keyframe nicht mehr in Relation zu dem zweiten Keyframe positioniert, sondern in die relative Postion zum ersten Keyframe gesetzt. Es besteht die Möglichkeit, die Koordinaten absolut zu vergeben. Hierfür muss diese Einstellung im Movement Track getätigt werden.[129] Des Weiteren kann jeder Keyframe als Bezier-, Linear-, oder konstanter Punkt definiert werden.[130] Es konnte festgestellt werden, dass es beim Laufen nötig ist, jeden Keyframe linear darzustellen, da der Charakter sonst beschleunigt und sich das „Schwimmen“[131] verstärkt.

Bei der Erstellung der Szene wurde festgestellt, dass Keyframes in Matinee nur eingeschränkt als Gruppe verschoben werden können. Animationsspuren lassen sich nur einzeln bewegen. Dies lässt nachträgliche Änderungen im Timing der Szene nur erschwert zu. Außerdem ist es nicht möglich, Keyframe-Gruppen effektiv zeitlich zu dehnen oder zu stauchen. Zudem ist es nicht möglich, Teile der Matinee fehlerfrei zu entfernen. Eine Funktion für diese Aufgabe ist zwar vorhanden. Jedoch ergab der Praxistest, dass bestimmte Spuren, wie die DirectorGroup[132] oder Anim Control Tracks bei dieser Funktion nicht entfernt werden können.

---

128 Vgl. http://udn.epicgames.com/Three/AdditiveAnimations.html (Entnahmedatum: 02.01.2012)

129 Vgl. http://udn.epicgames.com/Three/MatineeUserGuide.html#Creating A Matinee Action (Entnahmedatum: 02.01.2012)

130 Vgl. http://udn.epicgames.com/Three/CurveEditorUserGuide.html (Entnahmedatum: 02.01.2012)

131 Schwimmen: in diesem Zusammenhang: Die Schrittweite stimmt nicht mit dem Abstand der Movement-Keys überein und der Charakter läuft unnatürlich

132 DirectorGroup: Spur zum Umschalten zwischen verschiedenen Kameras

8.3.2 Anwendung von MorphWeights und FaceFX in Matinee

In der SkeletalMeshGroup lassen sich auch MorphWeight-Spuren und FaceFX-Spuren erstellen. Die MorphWeight-Spuren dienen dazu, Morph Targets anzusprechen. Für jedes Morph Target muss eine eigene Spur erstellt werden. Die Stärke kann mit Werten zwischen 0 und 1 mittels Keyframes zugewiesen werden.[133] Verschiedene MorphWeight-Spuren lassen sich kombinieren, allerdings addieren sich die MorphWeights. So wird beispielsweise die Mundöffnung bei einem Emotions-Morph-Target und einem Sprech-Morph-Target addiert, wenn beide gleichzeitig angewendet werden und der Mund öffnet sich stark.

Für FaceFX-Animationen ist pro FaceFX-AnimSet eine Spur nötig. Wie bei einem Anim Control Track wird hierbei eine Animation ausgewählt und via Keyframe im gewünschten Zeitraum abgespielt. Hierbei ließ sich festellen, dass es nicht möglich ist, diese Spuren zu kürzen oder zu zerteilen. Dies ist beim Import der Sounds zu beachten. Die SoundCues werden direkt mit der FaceFX-Animation abgespielt. Dadurch ist es nicht nötig, eine Audiospur zu erstellen und den Ton zu synchronisieren.[134] Die FaceFX-Animationen können in Matinee nicht direkt beeinflusst werden. Änderungen in Geschwindigkeit, der Phoneme oder der Stärke der Morph Targets müssen direkt in FaceFX-Studio vorgenommen werden. Es ist zu beachten, dass sich die Morph Targets bei FaceFX-Spuren, genau wie bei MorphWeight-Spuren auch, addieren. Ein Überschreiben ist nicht möglich. Es ist nicht möglich, mehrere FaceFX-Spuren für einen Charakter zu erstellen. Die vorhandene Spur wird dabei ausgeschaltet und nur die zweite Spur kann verwendet werden.

Bei der Erstellung des Films konnte man feststellen, dass bei der Lippensynchronisation von Sprache FaceFX gegenüber MorphWeights vorzuziehen ist. So ist die Lippensynchronisation genau auf die Sprache abgestimmt.

8.3.3 Kopfdrehung im Matinee

Für die Kopfdrehung ist eine HeadTracking-Spur von Nöten. HeadTracking ist die Ausrichtung eines Bones auf ein Ziel. Dieser kann mit Keyframes an- und ausgeschalten werden. Im Matinee ist hierbei noch auszuwählen, bis zu welcher maximalen Distanz

133 Vgl. http://udn.epicgames.com/Three/MorphTargets.html#Using Morph Targets in Matinee (Entnahmedatum: 02.01.2012)

134 Vgl. http://udn.epicgames.com/Three/IntroductionToFaceFX.html#Working with _FaceFX (Entnahmedatum: 02.01.2012)

der Charakter auf ein Ziel reagieren soll, welche Bones rotieren sollen und wie lange er ein Ziel maximal betrachtet.[135] Es ist möglich, mehrere HeadTracking-Spuren zu erstellen und so beispielsweise die Kopfdrehung getrennt von der Augenrotation zu anzusteuern. Allerdings ließ sich feststellen, dass nur ein Zielobjekt pro Szene möglich ist, da sich die gewählten Bones des Charakters immer auf den virtuellen Spieler ausrichten. Damit das Tracking für die Augen funktioniert und passende Ergebnisse liefert, muss das Ziel genügend Abstand zum Charakter haben. Ansonsten schielt der Charakter. Das HeadTracking kann nicht im Vorschaufenster wiedergeben werden, sondern lässt sich nur beim Abspielen der Sequenz überprüfen. Dies erschwert die Erstellung der HeadTracking-Spur.

8.4 Zielpunkt der Kopfdrehung

Bei der Erstellung der Testszenen zeigte sich, dass das Ziel der Kopfdrehung immer der Spieler ist, auch wenn es in Matinee die Möglichkeit gibt, andere Ziele, wie zum Beispiel Lichter oder Objekte auszuwählen. Jedoch rotiert der Bone immer zum Spieler. Dieser ist allerdings nicht direkt steuerbar. Lediglich der Spielstart ist variabel positionierbar. Eine Möglichkeit, die Kopfrotation trotzdem zu verwenden, ist das Erstellen eines Behälters, also einer Box mit offener Deckfläche wie in der nachfolgenden Abbildung dargestellt. Diese kann dann unter dem Spielstart platziert und in Matinee eingefügt werden. Der Box sollte eine Spur zur Steuerung der Sichtbarkeit und ein Movement Track zugewiesen werden. So kann sie als Element in Matinee gesetzt und gesteuert werden. Mit dem Movement Track kann dann die Bone-Rotation gesteuert werden. Bevor die Box bewegt wird, sollte sie mindestens zwei Sekunden still stehen, damit der virtuelle Spieler in die Box hinein fallen kann.

135 Vgl. http://udn.epicgames.com/Three/UsingSkeletalControllers.html#UDKSkelControl_LookAt / _UTSkelControl_LookAt (Entnahmedatum: 02.01.2012)

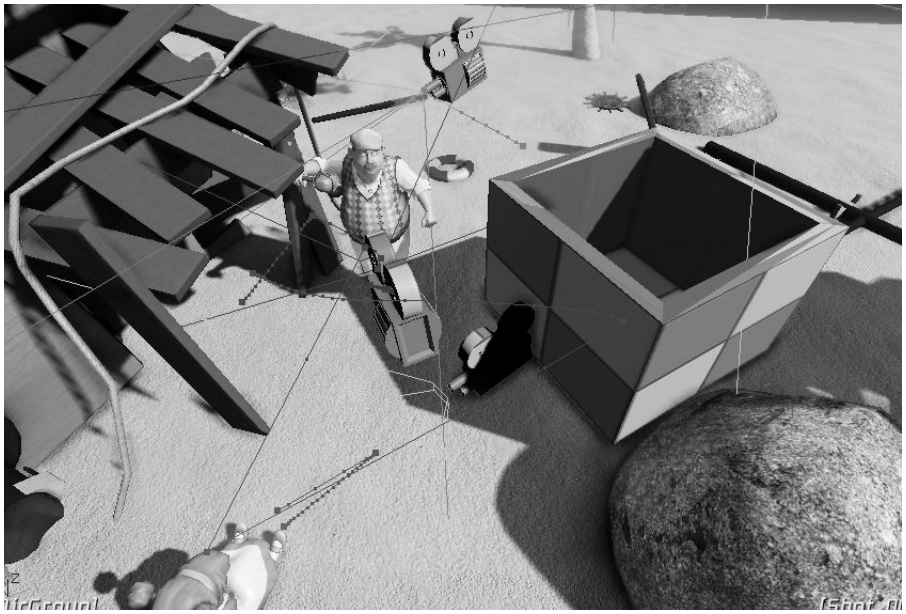

Abbildung 27: Box für das Ziel der Blickrichtung in der Szene (Eigene Darstellung, Charaktere von Pixable)

8.5 Steuerung von Kameras

Wenn im Level eine Kamera platziert worden ist, kann diese in Matinee angesprochen und kontrolliert werden. Hierbei können ein Movement Track und ein Field of View-Track (kurz: FOV) erstellt werden. Gesetzte Keyframes beim Movement Track sind basierend auf der Voreinstellung relativ, können aber auf absolute Werte gesetzt werden. Hierbei ist zu beachten, dass dies vor dem Setzen der Keyframes geschehen muss, da sich die Postion sonst ungewünscht verändert. Beim FOV-Track wird das Sichtfeld von 0° bis 360° eingestellt.[136] Diese Werte lassen sich mit der Brennweite eines Linsensystems vergleichen, welche sich mit folgender Formel in Relation setzen lassen:

$$\alpha = 2\arctan d\,\frac{d}{2\mathrm{f}}$$

(8.5-1) Formel zur Berechnung des Field of View

α entspricht hierbei dem FOV-Winkel und f steht für die Brennweite. Die Schärfentiefe wird im UDK nicht über die Kamera definiert. Außerdem ist es nicht möglich, die Blendenöffnung einzustellen. Kameras haben keinen verschiebbaren Zielpunkt. Das bedeutet, dass es nur möglich ist, die Kamera an ihrem Nullpunkt zu positionieren und zu rotieren.[137] Dies erschwert Kamerabewegungen, da zum Beispiel bei einem Schwenk der Zielpunkt immer neu gesucht werden muss. Für Kameras können eigene Post Pro-

136 Vgl. http://udn.epicgames.com/Three/MatineeUserGuide.html Entnahmedatum: 02.01.2012

cess-Einstellungen vorgenommen werden. So lässt sich zum Beispiel eine Farbkorrektur einer einzelnen Kamera vornehmen.[138] In den Einstellungen der Kamera wird auch das Bild-Seiten-Verhältnis direkt eingestellt.

8.6 Bildregie

Die Bildregie im UDK erfolgt über eine sogenannte DirectorGroup. Diese ist in jeder Matinee nötig, um sie abspielen zu können. In dieser DirectorGroup wird definiert, welche Kamera zu welchem Zeitpunkt gezeigt wird. In Abbildung 28 wird diese Gruppe mit verschiedenen Kameras dargestellt. Bleibt die Gruppe leer, wird automatisch aus Spieler-Perspektive wiedergegeben. Durch die Verwendung der DirectorGroup lässt sich ein großer Teil des Schnitts schon vor dem Rendering erledigen.[139] Wenn in eine Szene mehrere Kameras gesetzt werden und in der DirectorGroup zwischen ihnen gewechselt wird, lässt sich das Endergebnis des Films sehr gut previsualisieren.

Abbildung 28: DirectorGroup mit verschiedenen Kameras (Eigene Darstellung)

Weiche Blenden lassen sich in der DirectorGroup hardware-bedingt nicht definieren, da es nicht möglich wäre, diese zu rendern. Wenn diese gewünscht sind, müssen sie immer in einem Schnittprogramm realisiert werden.

137 Vgl. http://udn.epicgames.com/Three/CameraTechnicalGuide.html#Camera Animations (Entnahmedatum: 02.01.2012)

138 Vgl. http://udn.epicgames.com/Three/ColorGrading.html (Entnahmedatum: 02.01.2012)

139 Vgl. http://udn.epicgames.com/Three/MatineeUserGuide.html (Entnahmedatum: 02.01.2012)

9 Postproduktion

9.1 Post Process Effekte

Post Process Effekte sind Bild-Effekte, die dem Rendering mehr Qualität verleihen, aber dadurch auch die Rechenzeit erhöhen. Beispiele für solche Effekte sind Ambient Occlusion, also die Schatten, die durch Umgebungsverdeckung entstehen, Motion-Blur (Bewegungsunschärfe) und Bokeh-Effekte. Diese Effekte können im UDK entweder global angewandt, in einer Post-Process-Effekt-Kette vor die Elemente gesetzt, als Volumen auf einen bestimmten Bereich im Level begrenzt oder als Kamera-Einstellung, mit der die Post Prozesse mit neuen überschrieben werden.[140]

- **Ambient Occlusion:** Als Ambient Occlusion versteht man Umgebungsverdeckung. Dies ist ein Prozess, der die gegenseitige Verschattung an Objekten berechnet. Diese kann in die Lightmap gebacken werden und die Lichtqualität somit verbessert werden.[141] In Abbildung 29 ist erkennbar, dass die Szene rechts im Allgemeinen dunkler ist als links und vor allem in der Raumecke starke Schattierungen auftreten.

Abbildung 29: Ambient Occlusion ist aus (links) Ambient Occlusion ist an (rechts)[142]

140 Vgl. http://udn.epicgames.com/Three/PostProcessTechnicalGuide.html (Entnahmedatum: 02.01.2012)

141 Vgl. http://udn.epicgames.com/Three/PostProcessEffectReference.html#AmbientOcclusionEffect (Entnahmedatum: 02.01.2012)

142 http://www.chrisalbeluhn.com/UDK_Lightmass_Tutorial/20.jpg (Entnahmedatum: 02.01.2012)

- **Motion Blur:** Bewegungsunschärfe entsteht bei schnellen Bewegungen. Das Auge bzw. die Kamera kann den Positionsunterschied nicht wahrnehmen und das Einzelbild erscheint verschwommen. Motion Blur ist unbedingt nötig, um ein filmisches Ergebnis zu erzielen, da es Bewegungen für das Auge plausibler und optisch realistischer macht. [143]

Abbildung 30: kein Motion Blur (links) , Motion Blur (rechts) in der Unreal Engine[144]

- **Bokeh-Effekte:** Als Bokeh bezeichnet man den Effekt, wenn Kameras leuchtende Materialien, die unscharf sind, als n-eckige Flächen darstellen. Dies erzeugt einen realistischeren Look, da dieser Effekt bei realen Linsensystemen entsteht. Der Effekt lässt sich nur mit Unterstützung von DirectX 11 realisieren.[145]

Abbildung 31: Bokeh-Effekt in der Unreal Engine 3[146]

143 Vgl. Mental Ray for Maya, 3ds Max and XSI, Seite 259

144 http://udn.epicgames.com/Three/rsrc/Three/MotionBlur/camera_motionblur.jpg (Entnahmedatum: 02.01.2012)

145 http://udn.epicgames.com/Three/BokehDepthOfField.html (Entnahmedatum: 02.01.2012)

- **Depth of Field:** Schärfentiefe kann in der passenden Gelegenheit das Bild erheblich aufwerten, hierbei sollte sie aber immer bewusst als Mittel eingesetzt werden.

Abbildung 32: Depth of field[147]

- **Bloom:** Als Bloom bezeichnet man das Strahlen von hellen Flächen auf dunkleren Untergründen. Dies ist ein in Videospielen häufig sehr stark angewendeter Effekt. Für filmische Sequenzen ist er somit nur geeignet, wenn er nicht zu stark angewendet wird, da er sonst einen zu starken Videospiel-Look erzeugt.

- **Material Post Processes:** Materialien lassen sich so beispielsweise invertieren oder verändern.

- **Scene Effects:** Mit Szenen-Effekten lässt sich beispielsweise eine Farbkorrektur vornehmen. Im Praxistest wurde deutlich, dass es sich nicht anbietet, diese zu verwenden. Das Bild sollte in optimaler Qualität belassen und die Farbe in der Nachbearbeitung optimiert werden.

- **Anti-Aliasing:** Kantenglättung, die für die Reduktion von Treppeneffekten angwendet wird. Dies ist nötig, um ein Flackern im Film zu verhindern.

146 http://udn.epicgames.com/Three/BokehDepthOfField.html (Entnahmedatum: 02.01.2012)

147 http://cdn.gamerant.com/wp-content/uploads/Unreal-3-Next-Gen-Bokeh-Depth-Of-Field-2.jpg (Entnahmedatum: 02.01.2012)

9.2 Rendering

9.2.1 Rendering im UDK

Ein Rendering im UDK ist prinzipiell nicht zur Speicherung der Frames vorgesehen. Durch den gewöhnlichen Einsatz in Spielen ist die Darstellung der Frames nicht auf perfekte Einzelbilder ausgelegt, wie bei einem fotorealistischen Renderer, sondern darauf, schnell und effizient Grafik zu berechnen. Mit bestimmten Parametern lässt sich die maximale Qualität der Grafik-Engine trotzdem erzielen. Hierbei wird die Echtzeitdarstellung jedoch je nach verwendetem PC nicht mehr realisierbar. Im Gegensatz zu fotorealistischen Renderern ist die Zeitersparnis trotzdem noch enorm. Grundsätzlich gibt es zwei Möglichkeiten im UDK zu rendern:

- Das Exportieren des Films aus einer Matinee, (erfolgt als .avi oder Bitmap)
- Das Speichern von Screenshots, beim externen Abspielen des Levels (Bitmap)

Bei der Erstellung des Films direkt in Matinee können nur Bildrate und Auflösung eingestellt werden. Wenn die Ausgabe extern über Screenshots erfolgt, ist es nötig eine Verknüpfung zu erstellen, welche die Unreal Engine mit dem Level ausführt und dann Screenshots von jedes Einzelbild im Spiel erstellt. Hierbei erfolgt die Ausgabe immer als Bitmap-Folge. Jedoch kann man im Gegensatz zu Matinee verschiedene Parameter ausgeben und so das Endergebnis optisch optimal kontrollieren. Diese Einstellungen werden in die Verknüpfung eingetragen. Bei der Erstellung der Testszenen hat sich gezeigt, dass die Verwendung der externen Ausgabe gegenüber des Matinee Exports vorzuziehen ist, weil nur so eine hohe Kontrolle über die Bildqualität realisierbar ist.

Tabelle 2 zeigt, welche Parameter für die Erstellung der Screenshot-Folge nötig sind:

Befehl	**Bedeutung**
DUMPMOVIE	Befehl zur Erstellung einer Screenshot-Sequenz
BENCHMARK	Alle Frames werden vollständig und in optimaler Qualität berechnet
FPS	Frames pro Sekunde für den Benchmark
NOTEXTURESTREAMING	Texture-Streaming wird deaktiviert, um unscharfe Texturen zu verhindern
WINDOWED	Spiel wird im Fenster ausgeführt und benötigt eine Auflösung
FULLSCREEN	Spiel wird im Vollbild-Modus ausgeführt
ResX; ResY	Horizontale bzw. vertikale Auflösung für den Fenstermodus

Tabelle 4: Wichtige Kommandos für das Rendering.[148]

So ist es möglich, Screenshots als Bitmap-Sequenz zu erstellen Hierbei ist es auch unerheblich, ob der Renderprozess abbricht. Denn im Gegensatz zum Rendering mit konventionellen Programmen, wird beim Echtzeit-Rendering nicht an der abgebrochenen Stelle fortgesetzt, wenn ein Fehler entsteht.

9.2.2 Alternativen zum Echtzeit-Rendering

Szenen lassen sich als FBX- oder Collada-Dateien exportieren. Hierbei werden alle Movement-Tracks als Animationen übernommen. SkeletalMeshes werden als Null-Objekte importiert. Dies erlaubt ein Rendering in einer 3D-Bearbeitungssoftware.[149]

Der Test dieser Funktion hat ergeben, dass hierfür gewisse Anpassungen vorgenommen werden müssen. Die Materialien müssen im 3D-Bearbeitungsprogramm neu erstellt und zugewiesen werden. Die entstandenen Nullobjekte müssen gegen die 3D-Modelle getauscht werden und die Animationen müssen erneut zugewiesen werden. Terrain und Foliage werden nicht exportiert. Allerdings lässt sich im UDK eine Heightmap exportieren. So kann auch das Terrain optimal rekonstruiert werden.

Nach Meinung der Autoren wäre es so möglich, einen Export des Levels mit den Gamecast-Logfiles vorzunehmen und daraufhin die Szene im 3D-Barbeitungsprogramm

148 Vgl. http://udn.epicgames.com/Three/CommandLineArguments.html (Entnahmedatum: 02.01.2012)

149 Vgl. http://udn.epicgames.com/Three/MatineeUserGuide.html#Exporting and Importing Matinee Data (Entnahmedatum: 02.01.2012)

zu optimieren. So ist es realisierbar, ein optisch optimales Rendering bei Verwendung der Arbeitsabläufe mit der Games Engine und dem Gamecast-System zu erhalten. Allerdings benötigt dieser Arbeitsablauf einige Nachbereitung der Exportdaten, da Materialien nicht übernommen werden und SkeletalMeshes nur als Null-Objekte exportiert werden. Eine weitere mögliche Anwendung wäre es, die mit der Games Engine erzeugten Szenen zur Vorvisualisierung zu nutzen und das finale Rendering im 3D-Bearbeitungsprogramm durchzuführen.

9.2.3 Stereoskopie

Es besteht die Möglichkeit, den Film stereoskopisch in Echtzeit darzustellen. Hierbei kann mittels NVIDIA 3D Vision Direct[150] das Spiel mittels 3D-Shuttertechnik dargestellt werden. Es ist nicht möglich, diese Bilder als Bildfolge auszugeben.

Es hat sich gezeigt, dass die Ausgabe von stereoskopischen Bildmaterial umständlicher zu realisieren ist. Hierbei muss ein Stereo-Rig im UDK erzeugt werden. Das heißt, dass mittels eines Dummyobjekts mit zwei Sockeln die beiden Kameras befestigt werden und das Rig bewegt wird. Beide Kameras werden getrennt ausgegeben und in einem Compositing-Programm zu einem anaglyphen oder einem polarisierten Bild zusammengefügt. Dies geschieht je nach Programm mit vorgefertigten Effekten oder von Hand.

9.3 Postproduktion in einem Schnittsystem

Bei der Erstellung der Testszenen hat sich gezeigt, dass die Postproduktion der aufgenommenen Bilder nicht komplex ist. Hierbei besteht ein Unterschied, ob in der Director-Group bereits Schnitte zwischen den Kameras vorgenommen worden sind, oder ob alle Kameras über die volle Sequenzlänge gerendert wurden. Wenn bereits im UDK geschnitten wurde, ist es nur noch nötig, die fertigen Schnittsequenzen aneinander zu setzen und die gewünschten Effekte zu setzen. Wenn die Bilder der Kameras über die volle Länge berechnet wurden, empfiehlt sich ein Multi-Cam-Schnitt, wie ihn die meisten professionellen Schnittsysteme anbieten. Multi-Cam-Schnitt erlaubt es, mehrere Kameras simultan zu schneiden. So wird beispielsweise in Adobe Premiere[151] jede gewählte Kamera in einem Vorschaufenster dargestellt und es ist möglich, ähnlich wie bei

150 http://www.nvidia.de/object/3d-vision-main-de.html Entnahmedatum 02.01.2012

151 http://www.adobe.com/products/premiere.html?promoid=DJDTY (Entnahmedatum: 02.01.2012)

einem Bildmischer, diese zum geeigneten Zeitpunkt auszuwählen. So ist es möglich, Anschluss- oder Timing-Fehler zu vermeiden und einen schnellen Produktionsfluss zu garantieren. Hierbei empfiehlt es sich, dass der Schnittmeister sich bereits vor dem eigentlichen Schnitt eingehend mit der Schnittliste und dem Storyboard beschäftigt.

9.4 Vertonung

Es ist nicht möglich, im UDK Audiodaten zu den Bildern zu exportieren.[152] Abgesehen davon sind die Möglichkeiten im UDK zu vertonen eher eingeschränkt. Es lassen sich zwar beispielsweise interaktive Schrittgeräusche setzen, die nur abgespielt werden, wenn der Fuß auf die Erde setzt. Doch in Bezug auf Effekte, Mischung und Arbeitsablauf wurde bei der Bearbeitung der Testsequenz festgestellt, dass es deutlich sinnvoller ist, Sequenzen im Nachhinein zu vertonen.

152 Vgl. http://udn.epicgames.com/Three/MovieCapture.html#Audio capture (Entnahmedatum: 02.01.2012)

10 Qualitätsvergleich

10.1 Objekte und Charaktere

Im direkten Vergleich zu hochqualitativen Animationsfilmen müssen bei der Verwendung von Games Engines Abstriche gemacht werden. Diese belaufen sich auf die verwendeten 3D-Modelle, Texturen und Shader sowie die Berechnung von Licht und Schatten.

10.1.1 3D-Modelle

Durch die Verwendung von detailarmen 3D-Modellen wird die Erstellung von hochqualitativen 3D-Modellen stark eingeschränkt. Bei der Erstellung der Testszenen konnte folgendes festgestellt werden: Wenn bei einem Charakter ein Körperteil, wie zum Beispiel das Gesicht sehr detailliert ist, müssen die anderen Bereiche des Körpers mit weniger Details versehen werden. Ansonsten lässt sich das Modell nicht in der Games Engine verwenden, da die exportierbare Polygonzahl der Engine eingeschränkt ist. Daraus folgt, dass für eventuelle Close-Ups anderer Körperteile entweder detailliertere Modelle nötig sind oder mit dem detaillarmen 3D-Modell gearbeitet werden muss, was eine niedrigere optische Qualität zur Folge hat. Abbildung 33 zeigt, dass sich durch die Anwendung von Normal Maps die Detailarmut der Modelle gut verbergen lässt und eine hohe optische Qualität simuliert werden kann, die sich modernen Animationsfilmen annähert. Schwächen der Modelle sind meist nur in der Silhouette erkennbar.

Abbildung 33: Highpoly Objekt (oben), detailarmes Objekt ohne (Mitte) sowie mit Normal Map (unten) (Eigene Darstellung)

10.1.2 Haar- und Grassimulation

Detaillierte Haarsimulationen, wie beispielsweise bei „Sulley“ von „Die Monster AG“[153] sind innerhalb von Games Engines noch nicht möglich. Es ist nötig, die Haare als Texturen auf Polygone zu projizieren.[154] Diese Vereinfachung ist ein Kompromiss aus Qualität und Echtzeittauglichkeit. Komplexe Interaktionen mit diesen vereinfachten Haaren sind nicht möglich, da sich nur die Polygonstruktur der Haarflächen verformen lässt. Es lassen sich somit nur ganze Haargruppen beeinflussen. Diese komplexen Haarberechnungen werden in konventionellen Animationsfilmen auch zur Erstellung von Gras verwendet. In Games Engines ist hierbei auch die Vereinfachung über einzelne Polygonflächen nötig, die ihre Einflussbereiche durch Vertex Coloring zugewiesen bekommen.

10.1.3 Texturen und Materialien

Kompressionsbedingt gibt es Einschränkungen in der Texturqualität in Games Engines. Durch die Verwendung unkomprimierter Texturen steigt der Speicherbedarf stark an. Unkomprimierte Texturen sollten deshalb nur sparsam verwendet werden. Komplexe Materialien mit realistischen Reflexionen sind mit der Echtzeit-Engine nur schwer umzusetzen, da sie mit massiven Leistungseinbrüchen einhergehen. Die vereinfachten vorberechneten Reflexionen wirken sich allerdings nicht negativ auf den Gesamteindruck aus, wenn sie nicht bei stark spiegelnden Objekten angewendet werden. Auch im konventionellen Film können sehr diffuse Reflexionen, wie die von gebürstetem Metall, mittels Raytracing realistisch berechnet werden. Optische Unterschiede zu vorberechneten Reflexionen sind für den Betrachter dann meist nicht zu erkennen. Komplexe lichtdurchlässige Materialien, wie sie für menschliche Haut verwendet werden, können innerhalb von Games Engines und Raytracing-Renderern gleichermaßen eingesetzt werden.

10.1.4 Lichtberechnung

Durch die vordefinierte Schattenberechnung auf statischen Objekten ist die Unreal Engine weniger flexibel als 3D-Bearbeitungsprogramme wie zum Beispiel 3Ds Max oder Cinema 4D[155]. Tag- und Nachtwechsel können durch die fest in die Objekte einge-

153 http://www.imdb.de/title/tt0198781/ (Entnahmedatum: 02.01.2012)

154 http://udn.epicgames.com/Three/CreatingHairUsingAlpha.html (Entnahmedatum: 02.01.2012)

155 http://www.maxon.net/de/home.html (Entnahmedatum: 02.01.2012)

brannten Schatten nicht realisiert werden. Die Qualität der Schatten hängt stark von der verwendeten Lightmap-Auflösung ab. Bei besonders vielen Objekten mit hohen Lightmap-Auflösungen kann es unter Umständen zu einem Absturz bei der Vorberechnung der Lightmaps führen. Dies hat zur Folge, dass niedriger aufgelöste Lightmaps genutzt werden müssen. Hierdurch leidet die Qualität der Szene stark. Dynamische Schatten innerhalb des UDK werden mit Shadow Maps realisiert[156] Dadurch leidet allerdings die Performance und die Möglichkeit der Echtzeitanwendung wird eingeschränkt. Die Abbildung 34 zeigt mehrere Lightmapauflösungen auf einer 1024x1024 UU großen Fläche im Vergleich mit der Schattenberechnung des Mental Ray Renderers.

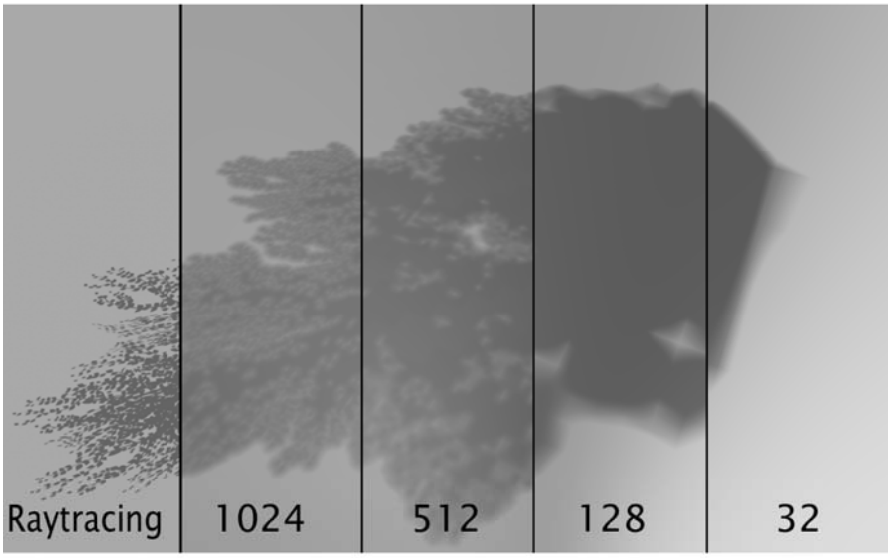

Abbildung 34: Diverse Lightmapauflösungen und Mental Ray Renderer (links) (Eigene Darstellung)

Im Vergleich zu Raytracingschatten, wie sie im Mental Ray Renderer möglich sind, ist die Qualität der Lightmapschatten geringer. Deutlich wirkt sich dies vor allem bei 3D-Modellen die Materialien mit Transparenzen verwenden aus.

10.1.5 Post Process Effekte

Nachträgliche Bildeffekte wie Schärfentiefe und Bewegungsunschärfe des UDK zeigen brauchbare Ergebnisse. Im Gegensatz zu konventionellen 3D-Bearbeitungsprogrammen sind die UDK-Effekte sehr ressourcenschonend. Problematisch ist die Schärfentiefe. Es ließ sich hierbei feststellen, dass die automatische Fokusanpassung auf Distanz teils sehr ruckartig erfolgt. Ein fester Depth of Field-Wert bringt gute Ergebnisse

156 Vgl. http://udn.epicgames.com/Three/LightingReference.html#Lightmaps (Entnahmedatum: 02.01.2012)

bei akzeptabler Bildqualität. Die einfachste Berechnungsmethode ist jedoch eine simpler Gaußscher Weichzeichner, der nicht im Fokus befindliche Bereiche einfach unscharf erscheinen lässt. Deshalb wirkt diese Methode unrealistisch. [157] Mit dem Bokeh Depth of Field lassen sich Ergebnisse erzielen, die mit den Effekten aktueller Animationsfilme vergleichbar sind. Hierfür sind Direct-X 11-Grafikkarten nötig. Die Depth of Field-Effekte sind nicht wie in Raytracing-Rendern abhängig von den Kameraeinstellungen. Bei diesen in konventionellen 3D-Bearbeitungsprogrammen eingesetzten Renderern haben Einstellungen wie die Brennweite einen Einfluss auf den Schärfebereich.[158] Im UDK wird dies lediglich über den Fokuspunkt sowie die Breite des Schärfebereichs geregelt.

Die Erstellung der Testszenen hat gezeigt, dass Anti Aliasing[159], also die Kantenglättung beim Rendering eine Schwäche im UDK ist. Es gibt zwei Möglichkeiten, die Verwendung des Full Scene Anti Aliasing[160] unter Direct-X 11 oder das Post Process Anti Aliasing[161]. Die erste Methode erlaubt es, alle Pixel einer Szene mit Anti Aliasing zu versehen. Die zweite Methode führt diese Berechnung nach dem eigentlichen Rendering durch und zeigt in Abhängigkeit ihrer Parameter Schwächen in kontraststarken Bereichen. Die Qualität des verwendeten Anti Aliasings unterscheidet sich im UDK von Grafikchip zu Grafikchip. Da jeder Hersteller andere Verfahren nutzt, können hier Qualitätsschwächen auftreten.[162] Da alle Grafikkarten, die im Test verwendet wurden (Geforce 9400M, Geforce GTS 250 und AMD HD4850) keine zufriedenstellenden Anti Aliasing Ergebnisse lieferten, war es nötig eine Alternative zu finden. Hierbei ließ sich feststellen, dass eine funktionierende Methode zum Anti Aliasing innerhalb des UDK das Rendern in sehr hohen Auflösungen ist. Nach dem Rendern kann das Bildmaterial wieder heruntergerechnet werden. Daraus folgt eine Minderung des Treppeneffektes.

10.2 Animationsqualität

Im UDK lassen sich Animationen gleichermaßen darstellen wie auch in 3D-Animationssoftware. Um eine hohe Animationsqualität zu ermöglichen, empfiehlt es sich, die komplette zu rendernde Szene bereits vor dem Import in das UDK zu animieren. Dies wur-

157 Vgl. http://udn.epicgames.com/Three/PostProcessEffectReference.html#BlurEffect (Entnahmedatum: 02.01.2012)

158 Vgl. Mastering Mental Ray – Rendering Techniques for 3D and CAD Pofessionals Seite 249

159 Vgl. Vray The Complete Guide 2nd Edition, Seite 167

160 Vgl. http://udn.epicgames.com/Three/AntiAliasingDX11.html (Entnahmedatum: 02.01.2012)

161 Vgl. http://udn.epicgames.com/Three/PostProcessAA.html (Entnahmedatum: 02.01.2012)

162 Vgl. http://udn.epicgames.com/Three/PostProcessAA.html (Entnahmedatum: 02.01.2012)

de bei der Erstellung der Testszenen festgestellt. Dies ergibt sich aus folgender Problematik: In das UDK importierte und auf einen Spieler ausgeführte Animationen werden immer wie in einem Computerspiel behandelt. Sie laufen kontinuierlich auf der Stelle ab. Positionsänderungen sind von der Spielerposition abhängig. Aus diesem Grund kann es vor allem bei komplexen Interaktionen mit der Umgebung, wie beispielsweise einem Laufen über eine Treppe oder das Springen über ein Hindernis, zu ungenauen Bewegungen führen, da im UDK nur die Position des Spielers definiert wird. Ein nachträgliches Bearbeiten der Animationen im UDK selbst ist nicht möglich. Dies lässt sich beheben, indem die komplette Szene bereits voranimiert wird. Dadurch ist es nicht mehr nötig, dass sich der Spieler innerhalb des UDK bewegt. Dadurch werden komplexe Interaktionen mit der Umgebung oder anderen Charakteren möglich. Auch lässt sich mehr Varianz in die Animationen bringen.

Da die Animationen immer in externen Programmen erstellt werden müssen, existiert das Risiko, dass Animationen beim Import falsch dargestellt werden. Dies ergibt sich beispielsweise dadurch, dass die Anzahl der Eckpunkte, die durch das Skinning verformt werden kann, im UDK beschränkt ist. In konventionellen Produktionen kann das Endergebnis immer sofort in der jeweiligen Animationssoftware betrachtet werden. Falls doch programmübergreifend gearbeitet wird, sind die bestehenden Pipelines darauf ausgelegt, die Dateien ohne Änderungen zwischen den Programmen zu tauschen, beispielsweise die Anbindung von Autodesk 3Ds Max zu Motionbuilder. Hierbei kann zur Animation das 3Ds Max-Skelett in Motionbuilder animiert und anschließend verlustfrei zu 3Ds Max exportiert werden. Dies ist bei der Arbeit im UDK nicht der Fall. Für zusätzliche Animationen des Charakterskeletts existiert im UDK keine Vorschau. Dies hat zur Folge, dass die Blickrichtung eines Charakters nicht in der 3D Vorschau angezeigt werden kann. Es gibt kein einheitliches Verfahren, um die ideale Positionierung des Zielpunktes für die Augen und den Kopf zu finden.

10.3 Filmische Mittel

Einstellungsgrößen können durch die frei steuerbaren Kameras realisiert werden. Allerdings sind Einstellungen wie Detail durch den niedrigen Detailgrad teilweise nur eingeschränkt realisierbar. Der Bildaufbau lässt sich frei und uneingeschränkt realisieren, da das Field of View nach Belieben geändert werden kann und die Kamera frei positioniert werden kann. Licht ist frei in den Szenen setzbar und klassische Beleuchtungssituationen lassen sich theoretisch problemlos umsetzen. Jedoch ist es problematisch, effektiv die Positionierung des Lichts zu verändern, da immer eine erneute Lichtberechnung er-

folgen muss. Innerhalb der Engine lässt sich Ton und Musik mit SoundCues zielgesteuert wiedergeben. Diese lassen sich beim Rendern allerdings nicht mit ausgeben. Das Material kann uneingeschränkt geschnitten werden und bietet in diesem Bereich die gleichen Möglichkeiten wie konventionelles Material auch.

10.4 Renderfehler

Die mit der Unreal Engine gerenderten Testszenen weisen in dunklen Bereichen teilweise deutlich sichtbare Bildfehler auf. Diese sind als farbige Verläufe sichtbar. In Abbildung 35 wird diese Problematik im rechten Bildrand erkennbar. Das Problem bestand beim Rendering bei drei getesteten Grafikkarten und konnte im Laufe der Arbeit nicht gelöst werden. Bei den Grafikchips handelte es sich um eine Geforce 9400M, eine Geforce GTS 250 und eine AMD HD4850. Eine mögliche Erklärung hierfür ist, dass sich die schwarzen Flecken und Farbverläufe auf die Ambient Occlusion-Berechnung innerhalb der Engine zurückführen lassen.

Abbildung 35: Renderfehler am rechten Bildrand (Eigene Darstellung)

10.5 Nachbearbeitungsmöglichkeiten

Einzelne Schichten des Bildes (Pässe) können im UDK nicht ausgegeben werden. So besteht weniger Raum für Nachbearbeitungen als bei konventionellen 3D-Animationsfilmproduktionen. In Renderern wie Mental Ray besteht die Möglichkeit, Szenen in mehreren Pässen zu rendern und diese nachträglich in Compositingsoftware wieder

zusammenzusetzen. Hierdurch hat der Künstler eine höhere Kontrolle über das Endergebnis. Es ist beispielsweise möglich, durch die Ausgabe von Objektmasken im Nachhinein ohne komplizierte Rotoskopie die Farbe eines Kleidungsstückes zu ändern. Die Ausgabe mehrerer Pässe ermöglicht beispielsweise das von LucasFilm entwickelte OpenEXR Bildformat.[163]

Mit dem UDK ist es allerdings auch möglich, spezielle Ausgabeshader zu nutzen, die es wiederum ermöglichen einen Tiefenpass auszugeben[164]. Mit diesem Tiefenpass ist es möglich, im Nachhinein Tiefenschärfe oder andere Effekte hinzuzufügen.[165] Daraus lässt sich herleiten, dass es problematisch ist, dass Szenen mit einem erneuten Durchlauf gerendert werden müssen. Dies hat zur Folge, dass nicht beeinflussbare Effekte wie Wind- oder Kleidungssimulationen sich anders verhalten als beim ersten Durchlauf. Daraus resultiert, dass sich die Renderpässe im Compositing nicht mehr fehlerfrei überlagern lassen.

163 Vgl. Mastering Mental Ray – Rendering Techniques for 3D and CAD Pofessionals Seite 67

164 Vgl. http://udn.epicgames.com/Three/PostProcessMaterials.html#SceneDepth Expression (Entnahmedatum: 02.01.2012)

165 Vgl. Mental Ray for Maya, 3ds Max and XSI, Seite 70

11 Fazit

Ziel der vorliegenden Arbeit war es, zu untersuchen, ob sich Filme mit ansprechender Optik und professionellem Aussehen mit einer Games Engine erzeugen lassen. Jeder Arbeitsschritt, der zur Erstellung eines Films nötig ist, wurde beleuchtet und auf seine Funktionalität in der Games Engine analysiert.

Grundsätzlich ist es mit Games Engines möglich, dramaturgische Erzählelemente optisch, mit filmischen Ausdrucksmitteln, umzusetzen. Die technischen Möglichkeiten hierfür sind zwar eingeschränkt aber es bleibt trotzdem genug Spielraum für das Erzielen von brauchbaren Ergebnissen. Die Einschränkungen lassen sich durch die Anpassung narrativer Methoden umgehen. Prinzipiell lassen sich alle Einstellungsgrößen umsetzen. Bei den Einstellungsgrößen Großaufnahme und Detail gibt es Einschränkungen durch den Detailgrad der Szenenobjekte. Kameraperspektiven und Bewegungen sind uneingeschränkt anwendbar. Ein ansprechender Bildaufbau lässt sich problemlos realisieren. Er ist aber durch die Vorberechnung der statischen Schatten eingeschränkt. Licht lässt sich zwar bewusst setzen, aber nur beschränkt im Prozess bearbeiten. Unschärfe lässt sich sowohl über die Kamera als auch über die Szenerie in Post Prozessen realisieren. Reißschwenks lassen sich mittels Motion Blur realisieren. Bei der Bildmontage lässt sich feststellen, dass nur harte Schnitte innerhalb von Matinee möglich sind. Für weiche Überblendungen benöitgen die Sequenzen die Bearbeitung in einem Schnittsystem.

Die Konzeption des Inhalts einer solchen Produktion erfordert genaue Planung, jedoch ließ sich feststellen, dass sich Änderungen in inhaltlichen Fragen ohne große Umwege durchführen lassen. Dies ergibt sich durch die niedrigen Renderzeiten sowie den modularen Aufbau der Engine. Charaktere mit gleichem Skelett können untereinander ausgetauscht werden, ohne dass neue Animationen erstellt werden müssen. Bei der Erstellung von Charakteren und Szenenobjekten ergaben sich die stärksten Einschränkungen. Gegenüber konventioneller Erstellungsmethoden benötigt die Detaileinschränkung durch die Modelle einen Ausgleich mittels Texturen und der weiterführenden Bearbeitung in der Engine. Bei der Animation von Charakteren ergab sich ein großer Nutzen, da loopbare Animationen wiederverwendet werden können und Animationen für mehrere Charaktere verwendet werden können. Jedoch ließ sich feststellen, dass der Nullpunkt von Objekten beim Export für Probleme sorgen kann, da die Unreal Engine immer die Hüfte eines Charakters als dessen Nullpunkt definiert. Dadurch können bei

verschiedenen Animationsprogrammen fehlerhafte Importe von Animationen entstehen. Weiterhin lassen sich geplottete Animationen nicht mehr effektiv bearbeiten. Interaktionen zwischen Charakteren werden durch den getrennten Export in die Engine erschwert, da mehrfach anwendbare Animationen nicht genau platziert werden können. Die größte Schwierigkeit in diesem Bereich ergibt sich also durch den Im- und Export von Animationsdaten. Bei Gesichtsanimationen kann festgestellt werden, dass sich mittels FaceFX eine gute Lippensynchronität der Sprache beim Charakter realisieren lässt. Die Verwendung von Morph Targets bietet einen schnellen und effizienten Arbeitsablauf. Die Stoffsimulation der Unreal Engine ist nach den Ergebnissen des Praxistests nur bedingt auf Charaktere anwendbar, da die Berechnung bei komplexen Interaktionen mit der Umgebung nicht vorhersehbar funktioniert. Die Erstellung der Szenerie lässt sich nach Erkenntnis der Autoren sehr gut für die Erstellung von Filmen verwenden. Die Landschaftserstellung bietet einen hohen Detailgrad, um ansprechende Settings zu realisieren. Bei der Erstellung der Testszenen hat sich ergeben, dass besonders Vegetation einfach und schnell realisierbar ist. Es ließ sich feststellen, dass die Weiterverarbeitung von Szenenobjekten teils mit viel Aufwand verbunden ist, da die Materialerstellung je nach Detailgrad aufwendiger ist als in konventionellen 3D-Bearbeitungsprogrammen. Die Physiksimulation von Umgebungsobjekten, wie zum Beispiel Bäumen im Wind, lässt sich schnell und einfach mit guten Ergebnissen realisieren. Die statische Lichtberechnung der Testszenen ist aufgrund der vorberechneten Beleuchtung ungenügend für die professionelle Erstellung eines Films. Um eine akzeptable Bild-Qualität zu erreichen, sind hohe Auflösungen der Lightmaps und höhere Vorberechnungszeiten vonnöten.

Das im Rahmen dieser Bachelorarbeit angewandte Gamecast-System ist noch nicht praxistauglich. Die Aufzeichnung von Szenen fällt positiv auf und ist bereits funktionstüchtig. Jedoch besteht noch Nachbesserungsbedarf in der Vereinfachung der aufgenommenen Daten. Die Emotionserkennung ist noch zu ungenau und ruckartig. Die MovementTracks der aufgenommenen Szenen sind nur bei hoher Datenrate brauchbar. Die Positionen der Spieler werden hierbei exakt gespeichert. Als problematisch erwies sich die enorme Anzahl der daraus resultierenden Keyframes, sowie das noch kaum vorhandene User Interface des Systems. Einfache Animationsaufzeichnungen, wie etwa die Aufzeichnung eines Videospiels, sind mit Gamecast möglich. Komplexe Filme, die eine Nachbearbeitung der Daten erfordern, lassen sich nach Auffassung der Autoren aufgrund der hohen Datendichte und ungenauen Emotionsdaten schwieriger realisieren.

Im Praxistest erwies sich Unreal Matinee als funktionstüchtig, um filmische Mittel zu realisieren. Die Steuerung der Kamera lässt klassische Einstellungsgrößen und ansprechenden Bildaufbau zu. Die Übersichtlichkeit bei komplexen Szenen mit vielen Akteuren ließ sich nur umständlich realisieren. Dies erschwert die Kontrolle über die Szene und behindert damit den Arbeitsablauf. Post Process Effekte werten das optische Ergebnis auf. Bewegungsunschärfe und Schärfentiefe bieten schnelle und gute Ergebnisse. Bei Ambient Occlusion ergab sich eine hohe Fehleranfälligkeit, da Bildfehler entstanden. Das Echtzeitrendering im UDK verkürzt die Renderzeit im Vergleich zu konventionellen Rendermethoden enorm und bietet dabei gute Ergebnisse. Auch das Rendering einer UDK-Szene lässt sich realisieren. Bei der Postproduktion in einem Schnittsystem wurden keine Probleme deutlich.

Zusammenfassend ist zu sagen, dass die Unreal-Engine im Verbund mit dem Gamecast-System noch nicht für die serielle Filmproduktion ausgereift ist, da Probleme wie das HeadTracking, die Stoffsimulation und Ambient-Occlusion-Fehler sichere Arbeitsabläufe verhindern.

Literaturverzeichnis

Jason Busby, Zak Parrish, Jeff Wilson: *Mastering Unreal Technology, Volume I: Introduction to Level Design with Unreal Engine 3*, Sams Publishing, Indianapolis, Indiana, USA, **2009**.

Jason Busby, Zak Parrish, Jeff Wilson: *Mastering Unreal Technology, Volume II: Advanced Level Design Concepts with Unreal Engine 3*, Sams Publishing, Indianapolis, Indiana, USA, **2009**.

Jennifer O´Conner: *Mastering Mental Ray - Rendering Tecniques for 3D & CAD Professionals,* Wiley Publishing, Inc., Indianapolis, Indiana, USA, **2010**.

Borko Furht: *Handbook of Multimedia for Digital Entertainment and Arts,* Springer Science+Business Media, LCC, New York, USA, **2009**.

Fabian Di Fiore, Bram Gerits und Frank Van Reeth: *Faking Dynamics of Cloth Animation for Animated Films,* Springer-Verlag Berlin Heidelberg, **2010**.

Boaz Livny: *Mental Ray for Maya, 3ds Max and XSI: A 3D Artist´s Guide to Rendering*, Wiley Publishing, Inc., Indianapolis, Indiana, USA, **2008**.

Francesco Legrenzi: VRay – The Complete Guide, Second Edition, Legrenzi Studio, Italien, **2010**

Jeremy Birn: Lightning & Rendering: 3D-Grafiken meisterhaft beleuchten, Addison-Wesley Verlag, München, Deutschland, **2007**.

Jason Gregory: *Game Engine Architecture*, Taylor and Francis Group, LCC, London, Großbritannien, **2009**.

Stefan Becker: *Virtuelle Welten mit der Raycasting-Technik darstellen*, c´t, Hannover, Deutschland 02/**1996**.

3ds Max 2011 Help, PDF, Autodesk, San Rafael, Kalifornien, USA, **2010**.

Unreal Development Network, URL: www.udn.com,Stand 02.01.2012

Workflow at Blue Sky Studio, URL: www.blueskystudios.com/content/process.php, Stand 02.01.2012.

Unreal Engine 3: Official Samaritian Demo, URL: http://www.youtube.com/watch?v=RSXyztq_0uM, Stand 02.01.2012

Anlagen

Anlage 1 Tabelle Gamecast Konverter

Zeit	50%Datendichte			33% Datendichte			3% Datendichte		
sec	**x**	**y**	**z**	**x**	**y**	**z**	**x**	**y**	**z**
12,1	1,37	2,01	0	1,38	2,02	0,01	0,09	0,15	0,01
12,2	1,27	1,85	0	1,26	1,83	0,07	3,93	5,73	0,19
12,3	1,2	1,78	0	1,16	1,68	0,1	6,74	9,82	0,33
12,4	1,11	3,62	0	1,08	1,5	0,25	9,21	13,41	0,45
12,5	1,03	1,52	0	0,99	1,6	0,05	11,65	16,96	0,57
12,6	0,91	1,33	0	0,86	1,23	0,1	14.16	20,61	0,57
12,7	0,45	0,66	0	0,41	0,59	0,18	16,61	24,18	0,81
12,8	0	0	0	0,23	0,34	0,25	10,74	15,64	0,59
12,9	2,42	0,84	0	1,78	2,57	0,07	0,57	0,92	0,03
13	0,03	0,88	0,5	1,65	0,93	0,04	0,67	0,96	0,28
13,1	0,96	1,39	0,19	0,01	0	0,12	0,29	0,44	0,29
13,2	0,49	0	0	0,49	0	0,15	12,98	18,18	0,25
13,3	1,05	1,52	0,06	1,44	2,1	0,3	23,85	34,73	0,89
13,4	0,06	5,91	0,22	0,16	6,23	0,02	34,85	44,73	1,52
13,5	0,01	0,05	0,01	3,76	2,5	0,05	47,53	69,28	1,9
13,6	0,12	0,08	0,15	0,74	1,09	0,02	81,41	79,2	2,33
13,7	0,6	0,88	0,91	1,17	2,22	0,03	53,82	75.33	2,55
13,8	0,01	0	0	0,39	0,55	0,05	42,59	61,95	1,57
13,9	0,07	0,09	0,21	0,77	1,11	0,15	26,27	38,25	2,41
14	2,03	2,96	0,31	0	0,01	0,35	15,39	22,42	0,93
14,1	2,71	0,01	0	0,01	0,01	0	0	0,01	0

Danksagung

Wir danken Herrn Professor Wierzbicki für die Betreuung und Erstkorrektur dieser Bachelorarbeit.

Außerdem danken wir Frau Klimant für die Zweitkorrektur dieser Arbeit.

Weiterhin möchten wir dem Forschungsprojekt Gamecast für die Inspiration und Unterstützung danken.

Großer Dank gilt dem Unternehmen Pixable für die Bereitstellung der Hauptcharaktere des im Rahmen dieser Bachelorarbeit entstandenen Films.

Martin Schuß gebührt großer Dank für die Vertonung des im Rahmen dieser Bachelorarbeit entstandenen Films und die stetige Unterstützung, die er uns entgegengebracht hat.

Eric Schubert dankt außerdem:

Ich möchte meiner Familie und Freunden für die emotionale Unterstützung bei der Erstellung dieser Bachelorarbeit danken.

Marika Obst dankt außerdem:

Ich danke meiner Familie für die Unterstützung und Zuversicht, die sie mir entgegengebracht haben. Weiterer Dank gebührt Ariane Brandenburg, die mich in jeder Phase der Arbeit unterstützt hat.

Eigenständigkeitserklärung

Hiermit erkläre ich, dass ich die vorliegende Arbeit selbstständig und nur unter Verwendung der angegebenen Literatur und Hilfsmittel angefertigt habe. Stellen, die wörtlich oder sinngemäß aus Quellen entnommen wurden, sind als solche kenntlich gemacht. Diese Arbeit wurde in gleicher oder ähnlicher Form noch keiner anderen Prüfungsbehörde vorgelegt.

Mittweida, den 16. Januar 2012 Marika Obst